▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación 19

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Informática y programación

19

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.

Gerente:
ANTONIO G. CUERPO.

Directora de producción:
MARIA LUISA SUAREZ PEREZ.

Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.

Diseño y maquetación:
BRAVO-LOFISH.

Fotografía:
EQUIPO GALATA.

Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomunicación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanera, Licenciada en Informática. APLICACIONES: Sociedad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (COBOL): Eloy Pérez, Licenciado en Informática. Ana Pastor, Licenciada en Informática.

Ediciones Siglo Cultural, S.A.

Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.

Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.

Distribución en Ecuador: Muñoz Hnos.

Distribución en Perú: DISELPESA.

Distribución en Chile: Alfa Ltda.

Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

ISBN del tomo: 84-7688-139-8
ISBN de la obra: 84-7688-068-7

Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.

Imprime:
MATEU CROMO. Pinto (Madrid).

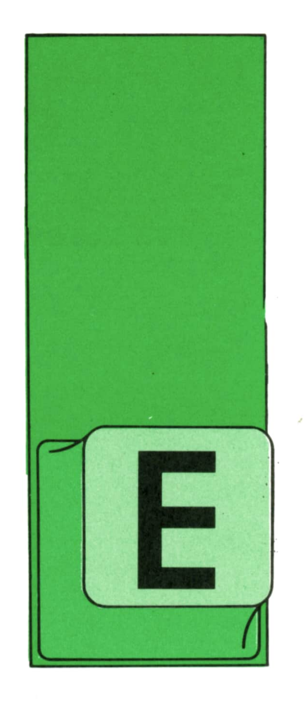
Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.

Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.

Julio, 1987.
P.V.P. Canarias: 335,–.

INDICE

BASIC





Bucles anidados

EN un mismo programa podemos utilizar varios bucles FOR-TO-NEXT, independientes o anidados. Los bucles anidados son aquellos que están dispuestos uno dentro de otro, por decirlo de algún modo.

Podemos anidar tantos bucles como deseemos, pero siempre debemos tener en cuenta que el orden para cerrar los bucles es inverso al de apertura, es decir, el primer bucle que abramos con la instrucción FOR-TO será el último que cerremos con NEXT y, por el contrario, el último que abramos será el primero que cerremos.

En la figura 1 podemos ver el esquema de dos bucles anidados de forma correcta.

Si estudiamos detenidamente este esquema podemos ver que por cada vez que se repite el bucle exterior (B), el interior (A) se ejecuta por completo. Por tanto, en el caso del esquema de la figura 1 por cada vez que se repite el bucle

```
BUCLE B ┌ FOR L=1 TO 3
        │  BUCLE A ┌ FOR M=2 TO 4
        │          │   .
        │          │   .
        │          └ NEXT M
        └ NEXT L
```

Esquema de bucles anidados.

B, el bucle A se repetirá tres veces, tomando los valores 2, 3 y 4.

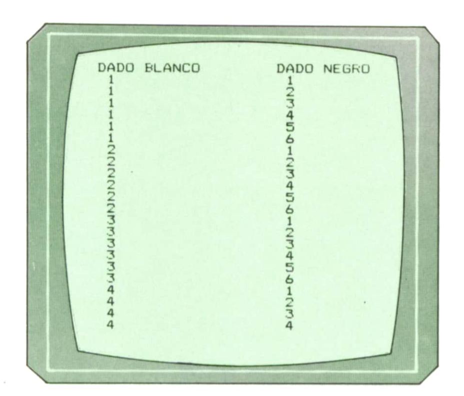
El programa 1 posee una estructura similar a la de la figura 1. Su objetivo es imprimir en pantalla todas las combinaciones posibles que podemos obtener al lanzar dos dados distintos, uno blanco y otro negro.

Podemos observar que el bucle denominado A está incluido en el interior del B.

El bucle exterior (líneas 60-100) sirve para imprimir en pantalla todos los posibles resultados del dado blanco, el bucle interior (líneas 70-90) se encarga de los resultados del dado negro.

```
10 REM ******************************
20 REM *  COMBINACIONES DE DADOS  *
30 REM ******************************
40 CLS
50 PRINT "DADO BLANCO";TAB(20);"DADO NEGRO"
60 FOR B=1 TO 6
70 FOR N=1 TO 6
80 PRINT B;TAB(20);N
90 NEXT N
100 NEXT B
110 PRINT "DADO BLANCO";TAB(20);"DADO NEGRO"
```

En la figura 2 podemos ver el aspecto de la pantalla durante la ejecución.




Presentación en pantalla del programa 1.

Evidentemente, todos los resultados no caben en la pantalla, por tanto, al finalizar la ejecución sólo podremos ver en pantalla los resultados finales. En cualquier caso, siempre podremos detener la ejecución antes de que finalice, en el momento que nos interese, utilizando las teclas que ya conocemos. En el SPECTRUM no sucede esto, ya que cuando se llena la pantalla la ejecución se detiene y aparece el mensaje *scroll?* para preguntarnos si deseamos que se imprima otra pantalla. En caso afirmativo no tenemos más que pulsar cualquier tecla (excepto N o BREAK).

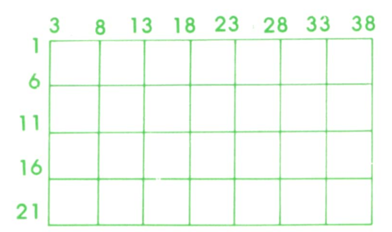
Los bucles anidados se utilizan mucho para realizar dibujos geométricos. Por ejemplo, el programa 2 tiene por objeto trazar en pantalla una retícula de asteriscos.

```
10 REM **************
20 REM *  RETICULA   *
30 REM **************
40 CLS
50 FOR C=3 TO 38 STEP 5
60 FOR F=1 TO 21
70 LOCATE F,C:PRINT "*"
80 NEXT F
90 NEXT C
100 FOR F=1 TO 21 STEP 5
110 FOR C=3 TO 38
120 LOCATE F,C:PRINT "*"
130 NEXT C
140 NEXT F
```

En este programa hemos utilizado dos parejas de bucles anidados. La primera (líneas 50-90) se encarga de imprimir en pantalla las líneas verticales, mientras que la segunda (líneas 100-140) traza las líneas horizontales.

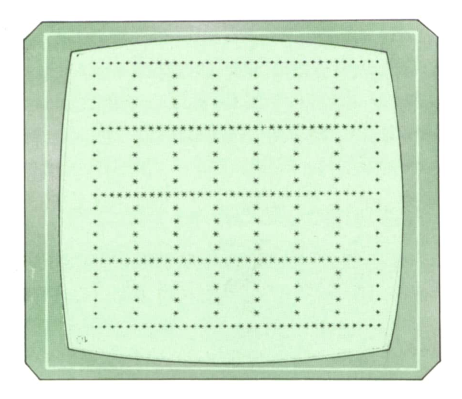
La variable C se utiliza para determinar la columna de la pantalla en la que se debe realizar la impresión en cada momento, mientras que la variable *f* controla las filas de la pantalla.

El objetivo es que la retícula se imprima en las filas y columnas indicadas en la figura 3, lo cual justifica los STEP 5 de los bucles exteriores.



Situación de la retícula en las filas y las columnas de la pantalla.

El aspecto final de la pantalla es el que podemos ver en la figura 4.




Presentación en pantalla del programa 2.

Debemos tener en cuenta que el programa 2 se ha desarrollado para una pantalla de 40 columnas como las del AMSTRAD, el IBM o los MSX. Sin embargo, si estamos utilizando un AMSTRAD debemos cambiar los LOCATE F,C que aparecen en las líneas 70 y 120 por LOCATE C,F, es decir, invertir el orden de *fila,columna* por *columna,fila*.

Por otra parte, si estamos utilizando un SPECTRUM tendremos que sustituir las siguientes líneas:

```
50 FOR C=1 TO 31 STEP 5
70 PRINT AT F,C;"*"
110 FOR C=1 TO 31
120 PRINT AT F,C;"*"
```

Además, podemos suprimir la línea 40. Por último, este programa no lo podría-

mos desarrollar de momento en el COMMODORE, ya que no dispone de la instrucción LOCATE ni de AT.

Veamos otro ejemplo de dibujo con bucles anidados. El programa 3 dibuja en el centro de la pantalla un triángulo rectángulo isósceles.

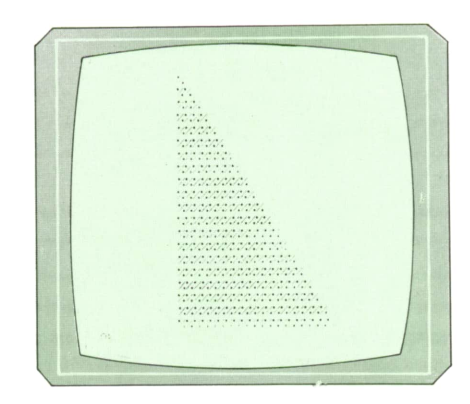
```
10 REM ***************************
20 REM *  TRIANGULO RECTANGULO    *
30 REM ***************************
40 CLS
50 FOR F=1 TO 20
60 FOR C=11 TO F+10
70 LOCATE F,C:PRINT "%"
80 NEXT C
90 NEXT F
```

En este caso el valor final del bucle interior depende del valor de la variable índice del bucle exterior. De este modo conseguimos que el número de signos % que se imprimen en cada fila de la pantalla vaya aumentando de uno en uno cada vez que se ejecuta el bucle exterior.

Una vez ejecutado el programa obtendremos una pantalla como la mostrada en la figura 5

Al igual que sucedía en el programa 2, aquí también tendremos que cambiar el LOCATE F,C de la línea 70 por LOCATE C,F, si trabajamos con un AMSTRAD.

En el SPECTRUM tendremos que sustituir las siguientes líneas:





Presentación en pantalla del programa 3.

```
60 FOR C=6 TO F+5
70 PRINT AT F,C;"%"
```

Como último ejemplo vamos a desarrollar el programa 4 que simula un reloj digital en el centro de la pantalla. Para ello utilizaremos cuatro bucles anidados.

```
10 REM *******************
20 REM *  RELOJ DIGITAL  *
30 REM *******************
40 CLS
50 FOR H=0 TO 23
60 FOR M=0 TO 59
70 FOR S=0 TO 59
80 LOCATE 13,17
90 PRINT H;":";M;":";S;"    "
100 FOR T=1 TO 1000:NEXT T
110 NEXT S
120 NEXT M
130 NEXT H
140 GOTO 50
```

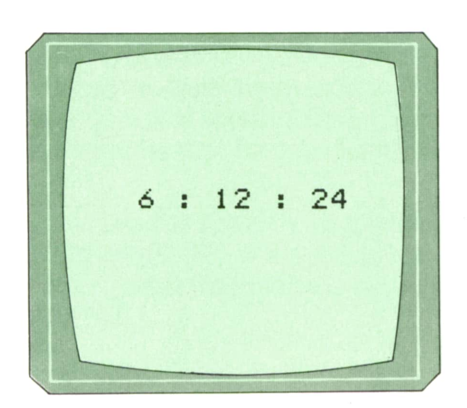
El primer bucle (líneas 50-130) es el encargado de determinar las horas, ya que al ser el más exterior es el que se repite más lentamente. A continuación el segundo bucle (líneas 60-120) controla los minutos y el tercero (líneas 70-110) los segundos. Lógicamente cuanto más dentro se sitúe un bucle anidado más veces se repetirá, por tanto, esta disposición es correcta.

Por otra parte, encontramos un cuarto bucle anidado en la línea 100 que parece un tanto extraño. La instrucción FOR-TO y la instrucción NEXT van seguidas, por tanto, no existe cuerpo del bucle. Esto es lo que se denomina un bucle de retardo y funciona como un temporizador. Si suprimimos la línea 100 del programa podemos observar que nuestro reloj va demasiado acelerado. El bucle de retardo hace que el reloj marche aproximadamente de segundo en segundo, ya que ese es el tiempo que emplea el ordenador en ejecutarlo.

Conviene advertir que el valor final del bucle de retardo varía en cada ordenador, ya que depende de la velocidad del procesador.

Finalmente la línea 50 produce un bucle infinito, ya que un reloj no se para a las doce de la noche, sino que vuelve a marcar las horas del nuevo día.

En la figura 6 podemos ver el aspecto de la pantalla durante la ejecución.




Presentación en pantalla del programa 4.

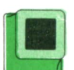
Recordemos que para que este programa funcione en el SPECTRUM tendremos que sustituir las líneas 80 y 90 por:

80 PRINT AT 12,12;H;":";M;":";S;" "

En el AMSTRAD sólo tenemos que cambiar el LOCATE 13,17 de la línea 80 por LOCATE 17,13, mientras que en el COMMODORE debemos suprimir la línea 80.

Finalmente, en algunos ordenadores como el AMSTRAD, el IBM o los MSX no es necesario especificar tantos NEXT como bucles anidados haya. Basta con indicar una sola instrucción NEXT con todas las variables índice a continuación, separadas por comas, pero siempre respetando el orden: la primera debe corresponder al último bucle abierto y la última al primer bucle. Por tanto, en el programa 4, por ejemplo, podríamos sustituir las líneas 110, 120 y 130 por:

110 NEXT S,M,H



Salidas y entradas

Para terminar con el tema de los bucles FOR-TO-NEXT vamos a hacer algunas consideraciones finales.

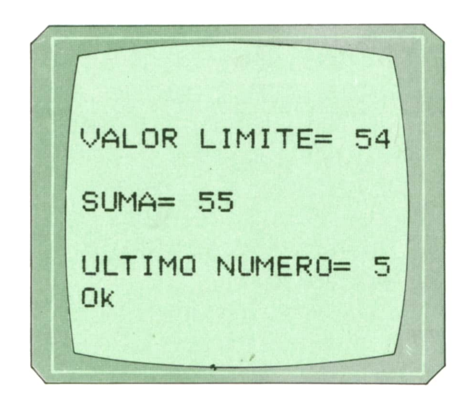
Podemos introducir en el interior de un bucle, anidado o no, una condición que suponga el abandono del mismo antes de que se haya ejecutado el número de veces que le corresponde según sus valores inicial y final y su incremento.

El programa 5 es un ejemplo de esta posibilidad. El objetivo es calcular la suma 12+22+32...hasta que se iguala o supera un valor introducido previamente.

```
10 REM **********
20 REM *  SUMA  *
30 REM **********
40 CLS
50 INPUT "VALOR LIMITE ";VL
60 IF VL<1 THEN GOTO 50
70 CLS
80 LET S=0
90 FOR I=1 TO VL
100 LET S=S+I*I
110 IF S>=VL THEN GOTO 130
120 NEXT I
130 PRINT "VALOR LIMITE=";VL
140 PRINT
150 PRINT "SUMA=";S
160 PRINT
170 PRINT "ULTIMO NUMERO=";I
```

Hemos establecido como valor final del bucle el almacenado en VL, sin embargo, este valor nunca se va a alcanzar, ya que en alguna de las ejecuciones del bucle se verificará la condición de la línea 110 y se saldrá a la línea 130.

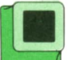
En la figura 7 podemos ver la presentación en pantalla tras una posible ejecución.



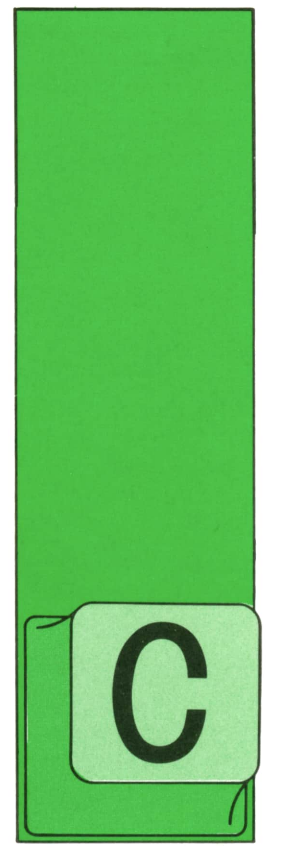

Presentación en pantalla del programa 5.

Por otra parte, si bien es posible salir de un bucle en cualquier momento antes de que finalice, no está permitido entrar en él sin pasar primero por la instrucción FOR-TO correspondiente que señala el inicio. Si tratamos de hacerlo aparecerá un mensaje de error del tipo *NEXT without FOR* que indica que el ordenador ha encontrado una instrucción NEXT sin haber pasado previamente por la instrucción FOR-TO correspondiente.

MAQUINA 8088



Rutinas



UANDO se trató de la estructura de los programas se dijo que éstos se pueden dividir en módulos, y los módulos en segmentos.

Pues bien, siguiendo esa línea, vamos a definir ahora las «rutinas» o «procedimientos» como divisiones de los segmentos de código. Las rutinas están constituidas por conjuntos de instrucciones destinados a realizar tareas auxiliares que pueden «llamarse» (es decir, se les puede transferir el control de ejecución) desde diferentes puntos de un programa. A su terminación, las rutinas devuelven el control de ejecución a la instrucción siguiente a la que realizó la «llamada».

Con este sistema se simplifican los programas, ya que las tareas que se repiten sólo tienen que programarse una vez, sustituyéndose el conjunto de instrucciones por una sola instrucción de llamada.

El mecanismo que permite que cuando se termina una rutina se vuelva a la instrucción adecuada, es el siguiente:

Cuando se ejecuta una instrucción de llamada, el 8088 carga en la pila de ejecución (STACK) la dirección a la que debe volver, bifurcando seguidamente al punto de entrada de la rutina definido en la instrucción. Inversamente, cuando se ejecuta una instrucción de fin de rutina, el 8088 descarga de la pila la dirección que allí se encuentre y bifurca a esa dirección.

Esta forma de funcionar hace necesario que las rutinas no alteren el estado de la pila, es decir, cuando terminan, las rutinas deben haber dejado la pila de ejecución en la misma situación en que la encontraron al comenzar. Esto no quiere decir que no puedan usar la pila, sino que deben utilizar tantas instrucciones de carga (PUSH y PUSHF) como de descarga (POP y POPF).

Aunque son tareas repetitivas, las rutinas admiten normalmente un cierto grado de variabilidad. Esto lo pueden conseguir utilizando parámetros, que son datos que se definen expresamente en cada llamada. Los métodos más usuales de definición de parámetros a las rutinas (lo que se denomina «paso de parámetros») son los siguientes:

— Copia de los parámetros en los registros.

— Carga de los parámetros en la pila mediante instrucciones PUSH, antes de realizar la llamada.

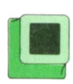
— Copia de los parámetros en un área de memoria que puede ser fija o variable. En caso de que dicha área sea variable, su dirección pueda darse por uno de los dos métodos anteriores.

Existen dos tipos de rutinas, las de tipo NEAR y las de tipo FAR. Las primeras son las que van a ser llamadas siempre desde el interior del segmento de código al que pertenecen. Las segundas pueden ser llamadas desde cualquier segmento del programa. Dicho tipo es una propiedad asociada a los puntos de entrada (la etiqueta por la que se le llama) y de salida (la instrucción de retorno), pero no

tiene relación con el resto de las instrucciones de la rutina.

Conviene aclarar que el conjunto de instrucciones que constituyen una rutina puede tener múltiples puntos de entrada y múltiples puntos de salida. Y debe cumplirse siempre que la instrucción de terminación sea del mismo tipo (NEAR o FAR) que la etiqueta por la que la rutina es llamada.

Los tipos de las rutinas se definen en las instrucciones PROC y los tipos de los puntos de entrada alternativos en las instrucciones LABEL.



Las instrucciones PROC y ENDP

Estas dos instrucciones no son ejecutables, es decir, no se traducen en códigos que deba ejecutar el 8088, sino que son seudo-operaciones directivas que sirven para acotar al ensamblador un bloque de instrucciones que constituye una rutina o procedimiento.

El formato de estas sentencias es el siguiente:

```
nombre-rutina PROC tipo
              .
              .
              .
nombre-rutina ENDP
```

nombre-rutina: Es el nombre que se le quiere asignar a la rutina. Dicho nombre debe coincidir exactamente en las instrucciones PROC y ENDP y es el que se utilizará en las llamadas.

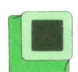
PROC y ENDP: Son los (seudo) códigos de operación. PROC indica el punto de entrada y es abreviatura de «procedure». ENDP significa «end of procedure» y sirve simplemente para marcar el final del bloque de instrucciones que constituyen la rutina.

La instrucción ejecutable que devuelve el control de ejecución a la instrucción siguiente a la de llamada es la instrucción RET que veremos más adelante.

tipo: Es un parámetro en el que se especifica el tipo de la rutina y puede ser NEAR o FAR.

NEAR significa «cerca» y se utiliza para las rutinas que van a ser llamadas siempre desde el mismo segmento de código al que pertenece la rutina.

FAR significa «lejos» y se utiliza para las rutinas que alguna vez tengan que ser llamadas desde un segmento de código diferente al de la rutina. Una rutina tipo FAR puede, por supuesto, ser llamada desde el mismo segmento al que pertenece.



La instrucción LABEL

Esta es una instrucción no ejecutable, que sirve para definir al ensamblador un nombre al que se le asocia un determinado tipo. Se puede usar tanto en los segmentos de código como en los de datos.

En los segmentos de código, los nombres así definidos pueden usarse en cualquier instrucción de bifurcación o de llamadas a rutina. Esta forma de definir etiquetas se usa frecuentemente para definir puntos de entrada alternativos en las rutinas, especialmente en las rutinas tipo FAR, ya que en las tipo NEAR se pueden usar etiquetas normales.

Su formato es el siguiente:

```
nombre    LABEL    tipo
```

nombre: Es el nombre que se quiere definir.

LABEL: Es el (seudo) código de operación y significa «etiqueta».

tipo: Es un parámetro en el que se especifica el tipo que se quiere asociar al nombre definido. El tipo puede ser NEAR o FAR con los mismos significados que se han explicado en la instrucción PROC.

Además, en los segmentos de datos, puede emplearse la instrucción LABEL para definir los tipos BYTE, WORD o DWORD. En esos casos, el nombre definido puede usarse como una variable de datos de uno, dos o cuatro bytes respectivamente.



Instrucciones CALL y RET

Son dos instrucciones ejecutables que sirven para comenzar y terminar la ejecución de rutinas.

Para ejecutar una rutina se emplea la instrucción CALL, que significa «llamada».

Para indicar que la rutina ha terminado y que el control de ejecución debe volver a la instrucción siguiente a la que la llamó se emplea la instrucción RET, abreviatura de «return», que significa «retorno».

Las instrucciones CALL y RET relacionadas con las rutinas tipo NEAR se denominan «intra-segmento» y sólo utilizan una palabra de la pila, que es el desplazamiento de la instrucción a la que tienen que volver.

Las instrucciones CALL y RET relacionadas con las rutinas tipo FAR se denominan «inter-segmentos» y necesitan utilizar dos palabras de la pila, que son el registro de segmento y el desplazamiento de la instrucción a la que tienen que volver.

El microprocesador debe distinguir las CALL y RET «intra-segmento» de las «inter-segmentos» y utiliza para ello códigos máquina diferentes. El programador, sin embargo, las escribe de la misma forma, y las distingue por el tipo asignado al nombre en las instrucciones PROC y LABEL.

Los formatos de estas instrucciones son los siguientes:

```
[etiqueta1:]    CALL    objetivo
[etiqueta2:]    RET     [descarga]
```

etiqueta1 y etiqueta2: Son nombres que se pueden utilizar opcionalmente para identificar a estas instrucciones.

CALL y RET. Son los códigos de operación.

objetivo: Es el operando por el que se define a la instrucción CALL cuál es la rutina que se quiere ejecutar. Este operando puede ser:

— La etiqueta de cualquier instrucción ejecutable del segmento. Esto provocará una llamada «intra-segmento».

— Un nombre de rutina definido en una instrucción PROC o un punto de entrada definido en una instrucción LABEL. La llamada será «intra-segmento» o «inter-segmentos», dependiendo de que se haya definido como tipo NEAR o FAR.

— Una referencia explícita a una posición de memoria (en cualquiera de las formas que se han explicado). En dicha posición se habrá cargado previamente la dirección de la rutina.

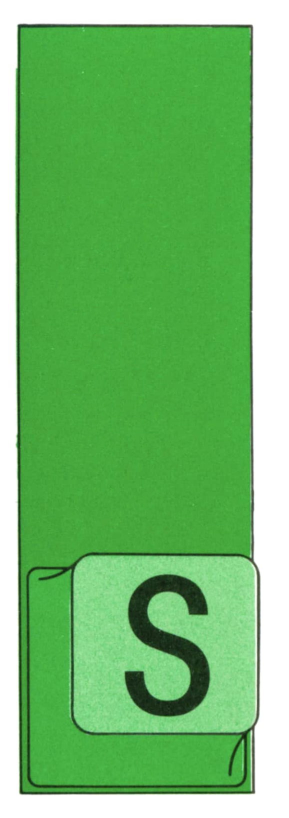
Si la posición de memoria referenciada estaba definida como WORD, se ejecuta una llamada «intra-segmento»; si estaba definida como DWORD, se ejecuta una llamada «inter-segmentos» y si se quiere realizar una llamada de tipo diferente al que le corresponde se debe anteponer uno de lo prefijos «WORD PTR» o «DWORD PTR», según se desee forzar que la llamada sea tipo «intra» o «inter-segmentos».

— Un registro AX, BX, CX, DX, SI, DI o BP en el cual se ha cargado previamente la dirección de la rutina. En este caso la llamada es siempre «intra-segmento».

descarga: Es un operando inmediato opcional de la instrucción RET que indica cuántos bytes deben descargarse de la pila inmediatamente después de ejecutarse la instrucción RET. Este operando es muy útil cuando se pasan parámetros a la rutina por el procedimiento de colocarlos en la pila antes de ejecutar la instrucción CALL. En ese caso la rutina debe terminar con la instrucción «RET n», donde *n* es un número igual al doble de las palabras que fueron colocadas en la pila.

PROGRAMAS

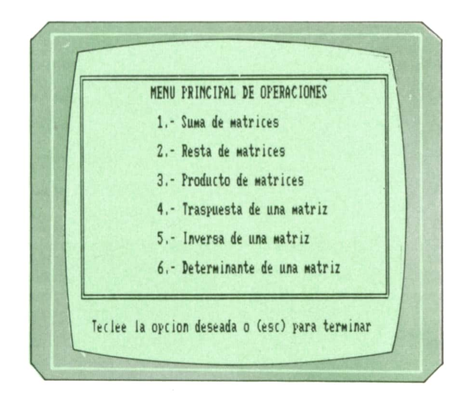
EDUCATIVOS • DE UTILIDAD • DE GESTION •DE JUEGOS



Programa: Operaciones con matrices

SIGUIENDO con los programas de matemáticas que hemos ido viendo en diferentes tomos, incluimos a continuación un programa que nos permitirá trabajar con matrices sin tener que realizar nosotros mismos los cálculos. Este programa nos permite hacer las siguientes operaciones:

— Sumar dos matrices.
— Restar dos matrices.
— Multiplicar dos matrices.
— Hallar la transpuesta de una matriz.
— Calcular la inversa de una matriz.
— Encontrar el determinante de una matriz.





Menú del programa.

El programa ha sido realizado en un IBM pc. Por ello puede funcionar en cualquier tipo de PC compatible que disponga de GWBASIC. Para el resto de los ordenadores las modificaciones son las siguientes:

SPECTRUM:

```
119 REM
120 REM
121 REM
122 REM
123 REM
124 PRINT AT 0,0;
128 PRINT AT 2,0;
130 PRINT AT 4,0;
132 PRINT AT 6,0;
134 PRINT AT 8,0;
136 PRINT AT 10,0;
138 PRINT AT 12,0;
139 PRINT "TECLEE LA OPCION (0 PARA TERMINAR)"
140 LET A$ = INKEY$
148 IF A$ = "0" THEN CLS: GOTO 9999
250 PRINT AT 10,0;
446 PRINT AT 2,0;
452 PRINT AT I + 3, K + 3;
463 PRINT AT 21,0;
465 LET A$ = INKEY$
```

También hay que borrar el grupo de líneas que van desde la 302 a la 325, ambas inclusive.

COMMODORE:

```
118 PRINT CHR$ (147)
119 REM
120 REM
121 REM
122 REM
123 REM
124 POKE 214,0: POKE 211,0
126 PRINT
128 PRINT
130 PRINT
132 PRINT
134 PRINT
136 PRINT
138 PRINT
139 PRINT "PULSE OPCION (0 PARA TERMINAR)"
140 GET A$
148 IF A$ = "0" THEN PRINT CHR$ (147): END
250 POKE 214,10: POKE 211,0
445 PRINT CHR$ (147)
446 POKE 214,1: POKE 211,0
452 POKE 214,I + 3: POKE 211, K + 3
463 POKE 214,23: POKE 211,0
465 GET A$
467 PRINT CHR$ (147)
```

También hay que borrar los números de línea comprendidos entre el 302 y 325.

AMSTRAD:

Sólo hay que cambiar de orden los argumentos de todas las sentencias LOCATE de forma que si nos aparece:

LOCATE 12,33

tendremos que poner:

LOCATE 33,12

Aparte de eso, también hay que realizar los siguientes cambios:

```
308 PRINT CHR$ (150);
310 PRINT CHR$ (154);
312 PRINT CHR$ (156)
315 PRINT CHR$ (149)
317 PRINT CHR$ (149)
320 PRINT CHR$ (147);
322 PRINT CHR$ (154);
324 PRINT CHR$ (153)
```

Se recomienda ejecutar este programa en el MODE 2 de pantalla.

MSX:

Para usar el programa en el MSX sólo hay que cambiar de orden los argumentos de todas las sentencias LOCATE que aparezcan en el programa. Por ejemplo, si vemos que en la línea 132 pone:

LOCATE 14,28

nosotros tendremos que poner:

LOCATE 28,14

Por otro lado, como el programa ha sido realizado para 80 columnas, en todas las líneas donde el segundo argumento (antes de darle la vuelta) sea mayor de 20, se recomienda dividir por dos y poner la parte entera. Así, la línea 130 que pone:

LOCATE, 12,28

pasaría a ser:

LOCATE 14,12

También hay que quitar las líneas que van desde la 119 a la 123, ambas inclusive, y desde la 302 a la 325, ambas inclusive.

```
100 REM ************************************************************************
101 REM *******           OPERACIONES CON MATRICES:                     ********
102 REM *******  SUMA,RESTA,PRODUCTO,INVERSA,TRANSPUESTA,DETERMINANTE   ********
103 REM ************************************************************************
104 REM
105 REM ************************************************************************
106 REM *******       AUTOR : JUAN MANUEL GUTIERREZ LEITON              ********
107 REM ************************************************************************
108 REM
109 REM ***********************************************************************
110 REM **********    (C.) EDICIONES SIGLO CULTURAL 1.987       ***************
111 REM ***********************************************************************
112 REM
113 DIM M(25,25)
114 DIM A(25,25)
115 DIM B(25,25)
116 DIM C(25,25)
117 DIM D(25)
118 CLS
119 LET F1=5
120 LET F2=20
121 LET C1=15
122 LET C2=65
123 GOSUB 303
124 LOCATE 6,25
125 PRINT "  MENU PRINCIPAL DE OPERACIONES  "
126 LOCATE 8,28
127 PRINT "1.- Suma de matrices"
128 LOCATE 10,28
129 PRINT "2.- Resta de matrices"
130 LOCATE 12,28
131 PRINT "3.- Producto de matrices"
132 LOCATE 14,28
```

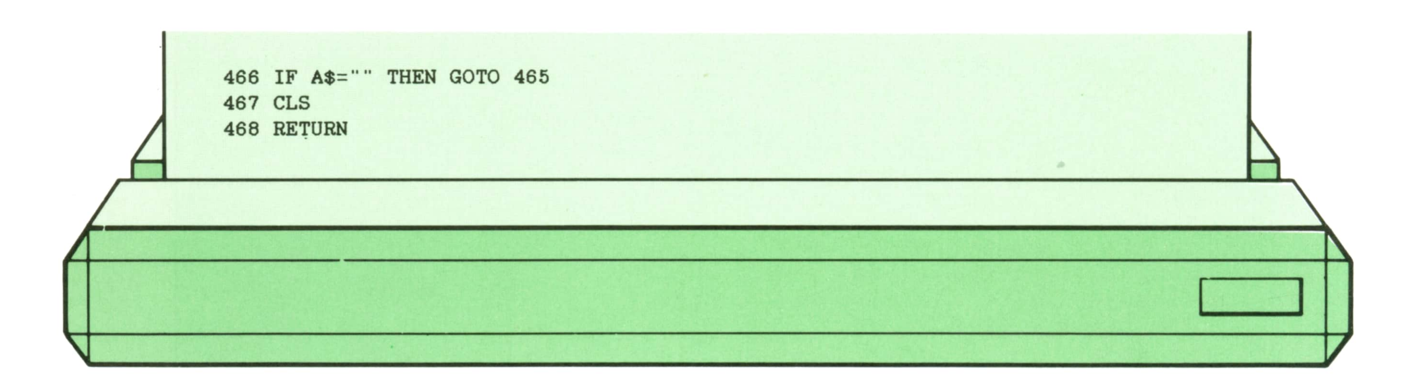
```
133 PRINT "4.- Traspuesta de una matriz"
134 LOCATE 16,28
135 PRINT "5.- Inversa de una matriz"
136 LOCATE 18,28
137 PRINT "6.- Determinante de una matriz"
138 LOCATE 22,17
139 PRINT "Teclee la opcion deseada o (esc) para terminar"
140 A$=INKEY$
141 IF A$="" THEN GOTO 140
142 IF A$="1" THEN GOTO 150
143 IF A$="2" THEN GOTO 167
144 IF A$="3" THEN GOTO 184
145 IF A$="4" THEN GOTO 201
146 IF A$="5" THEN GOTO 211
147 IF A$="6" THEN GOTO 239
148 IF ASC(A$)=27 THEN CLS:END
149 GOTO 140
150 GOSUB 362
151 LET NF=F1
152 LET NC=C1
153 GOSUB 378
154 IF ER=1 THEN GOTO 150
155 GOSUB 370
156 LET NF=F2
157 LET NC=C2
158 GOSUB 378
159 IF ER=1 THEN GOTO 155
160 IF F1<>F2 OR C1<>C2 THEN PRINT "no se pueden sumar matrices con distintas di
mensiones ":GOSUB 459:GOTO 118
161 GOSUB 327
162 GOSUB 339
163 GOSUB 256
164 GOSUB 441
165 GOSUB 459
166 GOTO 118
167 GOSUB 362
168 LET NF=F1
169 LET NC=C1
170 GOSUB 378
171 IF ER=1 THEN GOTO 167
172 GOSUB 370
173 LET NF=F2
174 LET NC=C2
175 GOSUB 378
176 IF ER=1 THEN GOTO 172
177 IF F1<>F2 OR C1<>C2 THEN PRINT "no se pueden restar matrices con diferentes
dimensiones ":GOSUB 459:GOTO 118
178 GOSUB 327
179 GOSUB 339
180 GOSUB 267
181 GOSUB 441
182 GOSUB 459
183 GOTO 118
184 GOSUB 362
185 LET NF=F1
186 LET NC=C1
187 GOSUB 378
188 IF ER=1 THEN GOTO 184
189 GOSUB 370
190 LET NF=F2
191 LET NC=C2
192 GOSUB 378
193 IF ER=1 THEN GOTO 189
194 IF C1<>F2 THEN PRINT "no se pueden multiplicar las matrices":GOSUB 459:GOTO
118
195 GOSUB 327
196 GOSUB 339
197 GOSUB 278
198 GOSUB 441
```

```
199 GOSUB 459
200 GOTO 118
201 GOSUB 362
202 LET NF=F1
203 LET NC=C1
204 GOSUB 378
205 IF ER=1 THEN GOTO 201
206 GOSUB 327
207 GOSUB 292
208 GOSUB 441
209 GOSUB 459
210 GOTO 118
211 GOSUB 362
212 LET NF=F1
213 LET NC=C1
214 GOSUB 378
215 IF ER=1 THEN GOTO 211
216 IF F1<>C1 THEN PRINT "no se puede calcular la matriz inversa de una matriz q
ue no sea cuadrada":GOSUB 459:GOTO 118
217 LET DI=F1
218 GOSUB 327
219 GOSUB 351
220 GOSUB 391
221 IF DE=0 THEN PRINT "la matriz no tiene inversa ya que el determinante es igu
al a cero":GOSUB 459:GOTO 118
222 LET X=DE
223 FOR F=1 TO DI
224    FOR C=1 TO DI
225       FOR I=1 TO DI
226          FOR J=1 TO DI
227             IF I=F AND J=C THEN LET M(I,J)=1:GOTO 230
228             IF I=F OR J=C THEN LET M(I,J)=0:GOTO 230
229             LET M(I,J)=A(I,J)
230          NEXT J
231       NEXT I
232       GOSUB 391
233       LET C(C,F)=(1/X)*DE
234    NEXT C
235 NEXT F
236 GOSUB 441
237 GOSUB 459
238 GOTO 118
239 GOSUB 362
240 LET NF=F1
241 LET NC=C1
242 GOSUB 378
243 IF ER=1 THEN GOTO 239
244 IF F1<>C1 THEN PRINT "no se puede calcular el determinante de una matriz que
 no sea cuadrada":GOSUB 459:GOTO 118
245 GOSUB 327
246 LET DI=F1
247 GOSUB 351
248 GOSUB 391
249 CLS
250 LOCATE 12,15
251 PRINT "el valor del determinante=";
252 PRINT DE
253 GOSUB 459
254 GOTO 118
255 REM
256 REM ******************************************
257 REM * subrutina para la suma de dos matrices *
258 REM ******************************************
259 REM
260 FOR I=1 TO F1
261    FOR J=1 TO C1
262       LET C(I,J)=A(I,J)+B(I,J)
263    NEXT J
264 NEXT I
```

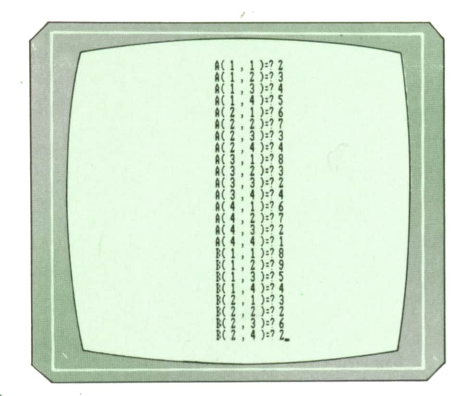
```
265 RETURN
266 REM
267 REM ********************************************
268 REM * subrutina para la resta de dos matrices *
269 REM ********************************************
270 REM
271 FOR I=1 TO F1
272    FOR J=1 TO C1
273       LET C(I,J)=A(I,J)-B(I,J)
274    NEXT J
275 NEXT I
276 RETURN
277 REM
278 REM ***********************************************
279 REM * subrutina para el producto de dos matrices *
280 REM ***********************************************
281 REM
282 FOR I=1 TO F1
283    FOR J=1 TO C2
284       LET C(I,J)=0
285       FOR K=1 TO C1
286          LET C(I,J)=C(I,J)+A(I,K)*B(K,J)
287       NEXT K
288    NEXT J
289 NEXT I
290 RETURN
291 REM
292 REM ********************************************************
293 REM * subrutina para calcular la traspuesta de una matriz *
294 REM ********************************************************
295 REM
296 FOR I=1 TO F1
297    FOR J=1 TO C1
298       LET C(J,I)=A(I,J)
299    NEXT J
300 NEXT I
301 RETURN
302 REM
303 REM ****************************************
304 REM * subrutina para el dibujo de ventanas *
305 REM ****************************************
306 REM
307 LOCATE F1,C1
308 PRINT CHR$(201);
309 FOR I=C1+1 TO C2-1
310    PRINT CHR$(205);
311 NEXT I
312 PRINT CHR$(187)
313 FOR I=F1+1 TO F2-1
314    LOCATE I,C1
315    PRINT CHR$(186)
316    LOCATE I,C2
317    PRINT CHR$(186)
318 NEXT I
319 LOCATE F2,C1
320 PRINT CHR$(200);
321 FOR I=C1+1 TO C2-1
322    PRINT CHR$(205);
323 NEXT I
324 PRINT CHR$(188)
325 RETURN
326 REM
327 REM ***********************************************
328 REM * subrutina de lectura de la matriz a(i,j) *
329 REM ***********************************************
330 REM
331 FOR I=1 TO F1
332    FOR J=1 TO C1
333       PRINT "A(";I;",";J;")=";
```

```
334         INPUT A(I,J)
335      NEXT J
336 NEXT I
337 RETURN
338 REM
339 REM ********************************************
340 REM * subrutina de lectura de la matriz b(i,j) *
341 REM ********************************************
342 REM
343 FOR I=1 TO F2
344     FOR J=1 TO C2
345         PRINT "B(";I;",";J;")=";
346         INPUT B(I,J)
347     NEXT J
348 NEXT I
349 RETURN
350 REM
351 REM ****************************************************
352 REM * subrutina de copia de la matriz a(i,j) en m(i,j) *
353 REM ****************************************************
354 REM
355 FOR I=1 TO F1
356     FOR J=1 TO C1
357         LET M(I,J)=A(I,J)
358     NEXT J
359 NEXT I
360 RETURN
361 REM
362 REM **********************************************************
363 REM * subrutina de peticion de las dimensiones de una matriz *
364 REM **********************************************************
365 CLS
366 INPUT "numero de filas de la matriz A ";F1
367 INPUT "numero de columnas de la matriz A ";C1
368 RETURN
369 REM
370 REM **************************************************
371 REM * subrutina de peticion de dimensiones de b(i,j) *
372 REM **************************************************
373 CLS
374 INPUT "numero de filas de la matriz B ";F2
375 INPUT "numero de columnas de la matriz B ";C2
376 RETURN
377 REM
378 REM *******************************************
379 REM * subrutina de errores en las dimensiones *
380 REM *******************************************
381 REM
382 IF NF<0 OR NC<0 THEN PRINT "la matriz no puede tener dimensiones menores que
 cero":GOTO 387
383 IF NF<>INT(NF) OR NC<>INT(NC) THEN PRINT "la matriz no puede tener dimension
es que sean numeros decimales ":GOTO 387
384 IF NF=0 OR NC=0 THEN PRINT "la matriz no puede tener ninguna dimension que s
ea igual a cero":GOTO 387
385 LET ER=0
386 GOTO 389
387 LET ER=1
388 GOSUB 459
389 RETURN
390 REM
391 REM *****************************************************************
392 REM *  subrutina para el calculo de los determinantes por el metodo *
393 REM * de triangulacion, es decir, todos  los elementos situados es- *
394 REM * trictamente bajo la diagonal principal son convertidos a cero *
395 REM *****************************************************************
396 REM
397 LET T=1
398 LET I=T
399 LET J=T
```

```
400 LET K=T
401 LET L=T
402 IF M(I,J)=0 THEN GOTO 430
403 LET D(T)=M(K,J)
404 LET I=T
405 LET D=M(K,I)
406 LET M(I,J)=M(I,J)/D
407 IF J=DI THEN GOTO 410
408 LET J=J+1
409 GOTO 406
410 LET I=I+1
411 LET J=T
412 LET RE=M(I,J)
413 LET M(I,J)=M(I,J)-M(K,J)*RE
414 IF J=DI THEN GOTO 417
415 LET J=J+1
416 GOTO 413
417 IF I=DI THEN 419
418 GOTO 410
419 IF T=DI-1 THEN GOTO 422
420 LET T=T+1
421 GOTO 398
422 LET T=DI
423 LET D(T)=M(DI,DI)
424 LET T=1
425 LET DE=1
426 FOR I=1 TO DI
427    LET DE=DE*D(I)
428 NEXT I
429 GOTO 439
430 LET I=I+1
431 IF M(I,J)=0 THEN GOTO 436
432 LET M(K,L)=M(K,L)+M(I,L)
433 LET L=L+1
434 IF L>DI THEN GOTO 403
435 GOTO 432
436 IF I=DI THEN GOTO 438
437 GOTO 430
438 LET DE=0
439 RETURN
440 REM
441 REM ***********************************************
442 REM * subrutina de impresion de la matriz c(i,j) *
443 REM ***********************************************
444 REM
445 CLS
446 LOCATE 2,15
447 PRINT "el resultado es:"
448 PRINT
449 FOR I=1 TO NF
450    LET K=1
451    FOR J=1 TO NC
452       LOCATE I+3,K+3
453       PRINT C(I,J)
454       LET K=K+9
455    NEXT J
456 NEXT I
457 RETURN
458 REM
459 REM ***********************************************
460 REM * subrutina de pulse una tecla para continuar *
461 REM ***********************************************
462 REM
463 LOCATE 23,25
464 PRINT "pulse una tecla para continuar"
465 A$=INKEY$
```

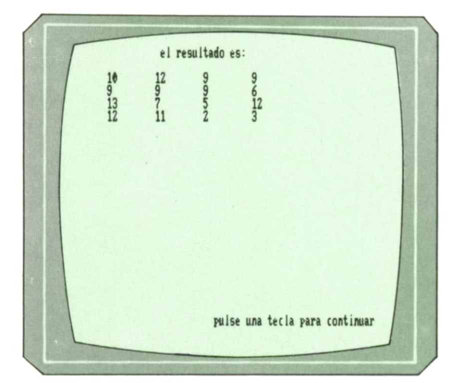


```
466 IF A$="" THEN GOTO 465
467 CLS
468 RETURN
```



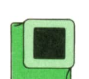


Dando valores a las matrices.





El resultado de sumar dos matrices.



Programa: Diseñador de sprites para MSX

El programa que vamos a ver a continuación nos permitirá definir nuestros propios SPRITES tanto en MSX como en el MSX2. El programa está capacitado para definir SPRITES de tamaño 8 × 8 ó 16 × 16. Mediante un menú al principio del programa se elige el tipo de SPRITE.

Por otro lado, a la hora de ver cómo aparece el SPRITE en la pantalla, también podemos elegir entre verlo normal o ampliado. Para ello disponemos de otro menú para poder elegir.

Una vez elegido el tipo de SPRITE que queremos definir, nos aparecerá una cuadrícula en la parte superior izquierda de la pantalla y será en ella donde tendremos que dibujar el SPRITE.

Para mover el cursor que parpadea dentro de la cuadrícula, utilizaremos las teclas del cursor. Cuando queramos encender un punto que esté apagado, nos colocaremos encima de él y pulsaremos la barra espaciadora. Si queremos borrar un punto que ya estuviese encendido, nos pondremos sobre él y usaremos, también, la barra espaciadora. Si queremos borrar un punto que ya estuviese encendido, nos pondremos sobre él y pulsaremos, también, la barra espaciadora.

Pulsando las teclas que van desde el F1 a la F8, podemos realizar algunas cosas con el contenido de la cuadrícula. El resumen de dichos comandos se encuentra permanentemente en las líneas inferiores de la pantalla.

```
1000 REM
1010 REM ************************
1020 REM * GENERADOR DE SPRITES *
1030 REM ************************
1040 REM
1050 SCREEN 0
```

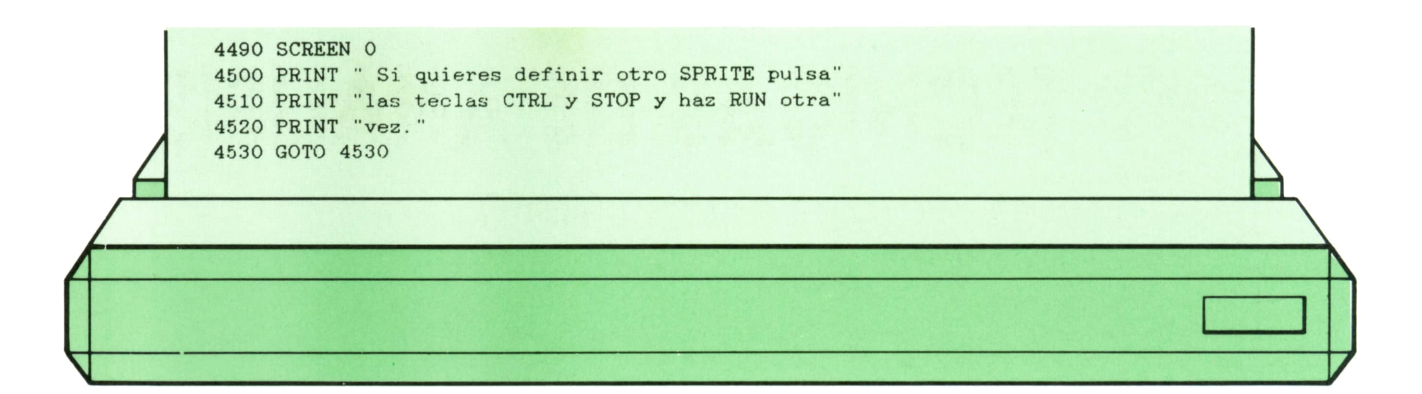
```
1060 KEY OFF
1070 CLEAR 600
1080 DEF FN I$(C$,P,B)=LEFT$(C$,TP-1)+CHR$(B)+RIGHT$(C$,32-TP)
1090 DEF FN C(A,B)=B+16*INT(A/8)+1
1100 DEF FN M(A$,A,B)=ASC(MID$(C$,FNC(A,B)))
1110 REM
1120 REM *******************
1130 REM * OPCION DE SPRITE *
1140 REM *******************
1150 REM
1160 COLOR 11,0,0
1170 CLS
1180 LOCATE ,,0
1190 PRINT"  GENERADOR DE SPRITES"
1200 PRINT" ======================"
1210 PRINT
1220 PRINT
1230 PRINT"Escoja tipo de SPRITE:"
1240 PRINT
1250 PRINT TAB(4);"1 -  8x8  Pixels."
1260 PRINT TAB(4);"2 - 16x16 Pixels."
1270 GOSUB 2270
1280 IF A$="1" THEN TA=7:L=TA ELSE TA=31:L=15
1290 PRINT
1300 PRINT "Escoja el tama$o del SPRITE:"
1310 PRINT
1320 PRINT TAB(4);"1 - Normal."
1330 PRINT TAB(4);"2 - Ampliado."
1340 GOSUB 2270
1350 PRINT
1360 PRINT
1370 PRINT "  Utiliza las  teclas del cursor paramoverte y  la barra  de espacio
  paradibujar/borrar un punto."
1380 PRINT
1390 PRINT "  Y PULSA UNA TECLA"
1400 A$=INKEY$
1410 IF A$="" THEN GOTO 1400
1420 SCREEN 1,VAL(A$)+1
1430 REM
1440 REM **********************
1450 REM * VARIABLE DONDE ESTA *
1460 REM * EL SPRITE          *
1470 REM **********************
1480 REM
1490 FOR I=0 TO 31
1500    C$=C$+CHR$(0)
1510 NEXT I
1520 SPRITE$(0)=C$
1530 PUT SPRITE 0,(180,28),15
1540 SI$=CHR$(219)
1550 NO$=CHR$(196)
1560 CLS
1570 LOCATE 19,2
1580 PRINT"Sprite:"
1590 CX=0
1600 CY=0
1610 DIM C(31)
1620 GOSUB 1830
1630 FC=0
1640 ON STRIG GOSUB 2020
1650 STRIG(FC) ON
1660 ON KEY GOSUB 2720,2350,3070,2880,3650,3470,3830,4430
1670 FOR G=1 TO 8
1680    KEY(G) ON
1690 NEXT G
1700 GOSUB 4310
1710 REM
1720 REM **********************
1730 REM * CONTROL DEL CURSOR *
```

```
1740 REM **********************
1750 REM
1760 M=STICK(FC)
1770 IF M=0 THEN GOTO 1760
1780 CX=(CX-(M=3)+(M=7) AND L)
1790 CY=(CY-(M=5)+(M=1) AND L)
1800 GOSUB 2170
1810 GOTO 1760
1820 REM
1830 REM ***************************
1840 REM * IMPRESION DE LA PARRILA *
1850 REM ***************************
1860 REM
1870 LOCATE 0,0,0
1880 FOR Y=0 TO TA
1890    LOCATE 8*INT(Y/16),Y-16*INT(Y/16)
1900    T=128
1910    T1=ASC(MID$(C$,Y+1))
1920    FOR X=0 TO 7
1930       IF (T1 AND T)=T THEN PRINT SI$; ELSE PRINT NO$;
1940       T=T/2
1950    NEXT X
1960 NEXT Y
1970 SPRITE$(0)=C$
1980 BEEP
1990 GOSUB 2170
2000 RETURN
2010 REM
2020 REM ***********
2030 REM * TRIGGER *
2040 REM ***********
2050 REM
2060 TP=FNC(CX,CY)
2070 KO=FNM(C$,CX,CY)
2080 KO=(KO XOR 2^(7-(CX AND  7)))
2090 C$=FNI$(C$,TP,KO)
2100 GOSUB 2150
2110 SPRITE$(0)=C$
2120 RETURN
2130 REM
2140 REM ***************************
2150 REM * POSICIONADOR DEL CURSOR *
2160 REM ***************************
2170 REM
2180 FOR CC=1 TO 40
2190 NEXT CC
2200 LOCATE CX,CY,0
2210 PA=FNM(C$,CX,CY)
2220 IF (PA AND 2^(7-(CX AND 7))) THEN PRINT SI$; ELSE PRINT NO$;
2230 LOCATE CX,CY,1
2240 RETURN
2250 REM
2260 REM *********************
2270 REM * ESPERA PULSACION *
2280 REM *********************
2290 REM
2300 A$=INKEY$
2310 IF A$<>"1" AND A$<>"2" THEN 2300
2320 RETURN
2330 REM
2340 REM **********************
2350 REM * ROTACION IZQUIERDA *
2360 REM **********************
2370 REM
2380 GOSUB 2510
2390 IF TA=31 THEN 2440
2400 FOR I=0 TO 7
2410    C(I)=(C(I)*2 AND 254)+(C(I) AND 128)/128
2420 NEXT I
```

```
2430 GOTO 2620
2440 FOR I=0 TO 15
2450    T1=(C(I) AND 128)/128:T2=(C(I+16) AND 128)/128
2460    C(I)=(C(I)*2 AND 254)+T2:C(I+16)=(C(I+16)*2  AND 254)+T1
2470 NEXT I
2480 GOTO 2620
2490 REM
2500 REM ***********************
2510 REM * PASA LA MATRIZ C$ A *
2520 REM * LA MATRIZ C PARA EL *
2530 REM * CALCULO NUMERICO.   *
2540 REM ***********************
2550 REM
2560 FOR I=0 TO 31
2570    C(I)=ASC(MID$(C$,I+1,1))
2580 NEXT I
2590 RETURN
2600 REM
2610 REM ***************************
2620 REM * SALIDA DE LAS RUTINAS   *
2630 REM * DE ROTACIONES Y ESPEJOS *
2640 REM ***************************
2650 REM
2660 C$="":FOR I=0 TO 31:C$=C$+CHR$(C(I)):NEXT
2670 SPRITE$(0)=C$
2680 GOSUB 1830
2690 RETURN
2700 REM
2710 REM ********************
2720 REM * ROTACION DERECHA *
2730 REM ********************
2740 REM
2750 GOSUB 2510
2760 IF TA=31 THEN 2810
2770 FOR I=0 TO 7
2780    C(I)=C(I)/2+128*(C(I) AND 1)
2790 NEXT I
2800 GOTO 2620
2810 FOR I=0 TO 15
2820    T1=128*(C(I) AND 1):T2=128*(C(I+16) AND 1)
2830    C(I)=C(I)/2+T2:C(I+16)=C(I+16)/2+T1
2840 NEXT I
2850 GOTO 2620
2860 REM
2870 REM *******************
2880 REM * ESPEJO VERTICAL *
2890 REM *******************
2900 REM
2910 GOSUB 2510
2920 IF TA=31 THEN 2970
2930 FOR G=0 TO 3
2940    SWAP C(G),C(7-G)
2950 NEXT G
2960 GOTO 2620
2970 FOR G=0 TO 7
2980    SWAP C(G),C(15-G)
2990    SWAP C(G+16),C(31-G)
3000 NEXT G
3010 GOTO 2620
3020 REM
3030 REM *********************
3040 REM * ESPEJO HORIZONTAL *
3050 REM *********************
3060 REM
3070 GOSUB 2510
3080 IF TA=31 THEN 3180
3090 FOR G=0 TO 7
3100    T1=C(G)
3110    C(G)=0
```

```
3120    FOR I=0 TO 3
3130       C(G)=C(G)-(2^I)*((2^(7-I) AND T1)<>0)
3140       C(G)=C(G)-(2^(7-I))*((2^I AND T1)<>0)
3150    NEXT I
3160 NEXT G
3170 GOTO 2620
3180 FOR G=0 TO 15
3190    T1=C(G)
3200    T2=C(G+16)
3210    C(G)=0
3220    C(G+16)=0
3230    FOR I=0 TO 7
3240       C(G)=C(G)-(2^I)*((2^(7-I) AND T2)<>0)
3250       C(G+16)=C(G+16)-(2^(7-I))*((2^I AND T1)<>0)
3260    NEXT I
3270 NEXT G
3280 GOTO 2620
3290 REM
3300 REM ********
3310 REM * LOAD *
3320 REM ********
3330 REM
3340 GOSUB 3370
3350 REM
3360 REM *****************
3370 REM * PREPARA INPUT *
3380 REM *****************
3390 REM
3400 POKE &HF3F9,PEEK(&HF3FB)
3410 POKE &HF3F8,PEEK(&HF3FA)
3420 LOCATE 23,29
3430 PRINT STRING$(232,127)
3440 RETURN
3450 REM
3460 REM ********
3470 REM * SAVE *
3480 REM ********
3490 REM
3500 GOSUB 3370
3510 GOSUB 4140
3520 GOSUB 4230
3530 PRINT"    Y PULSE UNA TECLA";
3540 A$=INKEY$
3550 IF A$="" THEN 3540
3560 OPEN "cas:"+N$ FOR OUTPUT AS #1
3570 FOR G=0 TO TA
3580    PRINT#1,ASC(MID$(C$,G+1,1))
3590 NEXT
3600 CLOSE#1
3610 GOSUB 4310
3620 RETURN
3630 REM
3640 REM ********
3650 REM * LOAD *
3660 REM ********
3670 REM
3680 GOSUB 3370
3690 GOSUB 4140
3700 GOSUB 4230
3710 C$=""
3720 OPEN "cas:"+N$ FOR INPUT AS #1
3730 IF EOF(1) THEN 3770
3740 INPUT#1,A
3750 C$=C$+CHR$(A)
3760 GOTO 3730
3770 CLOSE#1
3780 GOSUB 1830
3790 GOSUB 4310
3800 RETURN
```

```
3810 REM
3820 REM ***************************
3830 REM * SAVE COMO LINEA DE DATA *
3840 REM ***************************
3850 REM
3860 GOSUB 3370
3870 GOSUB 4140
3880 INPUT "Numero de linea = ";LI
3890 GOSUB 4230
3900 PRINT"       Y PULSE UNA TECLA";
3910 A$=INKEY$
3920 IF A$="" THEN GOTO 3910
3930 OPEN "cas:"+N$ FOROUTPUT AS #1
3940 A$=STR$(LI)+" DATA "
3950 FOR G=1 TO TA
3960    A$=A$+"&H"+HEX$(ASC(MID$(C$,G,1)))+","
3970 NEXT
3980 A$=A$+"&H"+HEX$(ASC(MID$(C$,TA+1,1)))
3990 PRINT#1,A$
4000 CLOSE#1
4010 GOSUB 3360
4020 LOCATE 0,17,0
4030 PRINT "Esta es la linea grabada:"
4040 PRINT
4050 PRINT A$
4060 PRINT
4070 PRINT "    Y PULSA UNA TECLA"
4080 A$=INKEY$
4090 IF A$="" THEN GOTO 4080
4100 GOSUB 4310
4110 RETURN
4120 REM
4130 REM ******************
4140 REM * PREGUNTA NOMBRE *
4150 REM ******************
4160 REM
4170 LOCATE 0,17
4180 INPUT "Nombre";N$
4190 IF N$="" THEN 4170
4200 RETURN
4210 REM
4220 REM ***********************
4230 REM * MENSAJES DE CASSETTE *
4240 REM ***********************
4250 REM
4260 LOCATE 0,22,0
4270 PRINT" Ponga en marcha el cassette"
4280 RETURN
4290 REM
4300 REM *************
4310 REM * PRINT MENU *
4320 REM *************
4330 REM
4340 LOCATE 29,23,0
4350 PRINT STRING$(232,127)
4360 LOCATE 0,16
4370 PRINT"Opciones:"
4380 PRINT"F1-Rota der.","F5-Load","F2-Rota izq.","F6-Save norm.","F3-Espejo hor
.F7-Save data.","F4-Espejo ver.F8-Stop."
4390 GOSUB 2150
4400 RETURN
4410 REM
4420 REM ********
4430 REM * STOP *
4440 REM ********
4450 REM
4460 REM
4470 SPRITE$(0)=CHR$(0)
4480 CLS
```



```
4490 SCREEN 0
4500 PRINT " Si quieres definir otro SPRITE pulsa"
4510 PRINT "las teclas CTRL y STOP y haz RUN otra"
4520 PRINT "vez."
4530 GOTO 4530
```

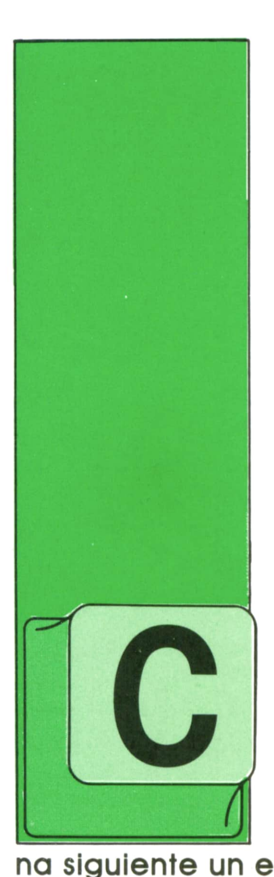
Una vez que tengamos definido el SPRITE, podemos guardarlo en cinta de casete de dos maneras distintas. La primera de ellas nos almacena el SPRITE con forma de línea DATA, de manera que cuando necesitemos utilizar dicho SPRITE para un programa que nosotros estemos realizando, sólo tenemos que hacer un MERGE entre el programa y la línea que contiene la definición del SPRITE.

La otra manera de almacenarlo es una forma interna de manera que el programa pueda más tarde volver a leer dicho SPRITE para que el usuario pueda modificarlo si lo desea. Cuando el usuario necesite leer un SPRITE que tiene grabado en cinta, tendrá que usar la opción LOAD del menú general.

Este programa, como ya habrá visto el lector al usuario, sólo nos permite definir un SPRITE cada vez. Esto, aunque pueda parecer una molestia, en el fondo no lo es tanto y hace que el programa sea más corto y fácil de entender.

TECNICAS DE ANALISIS

CONTENIDO DE UN DOCUMENTO DE SALIDA



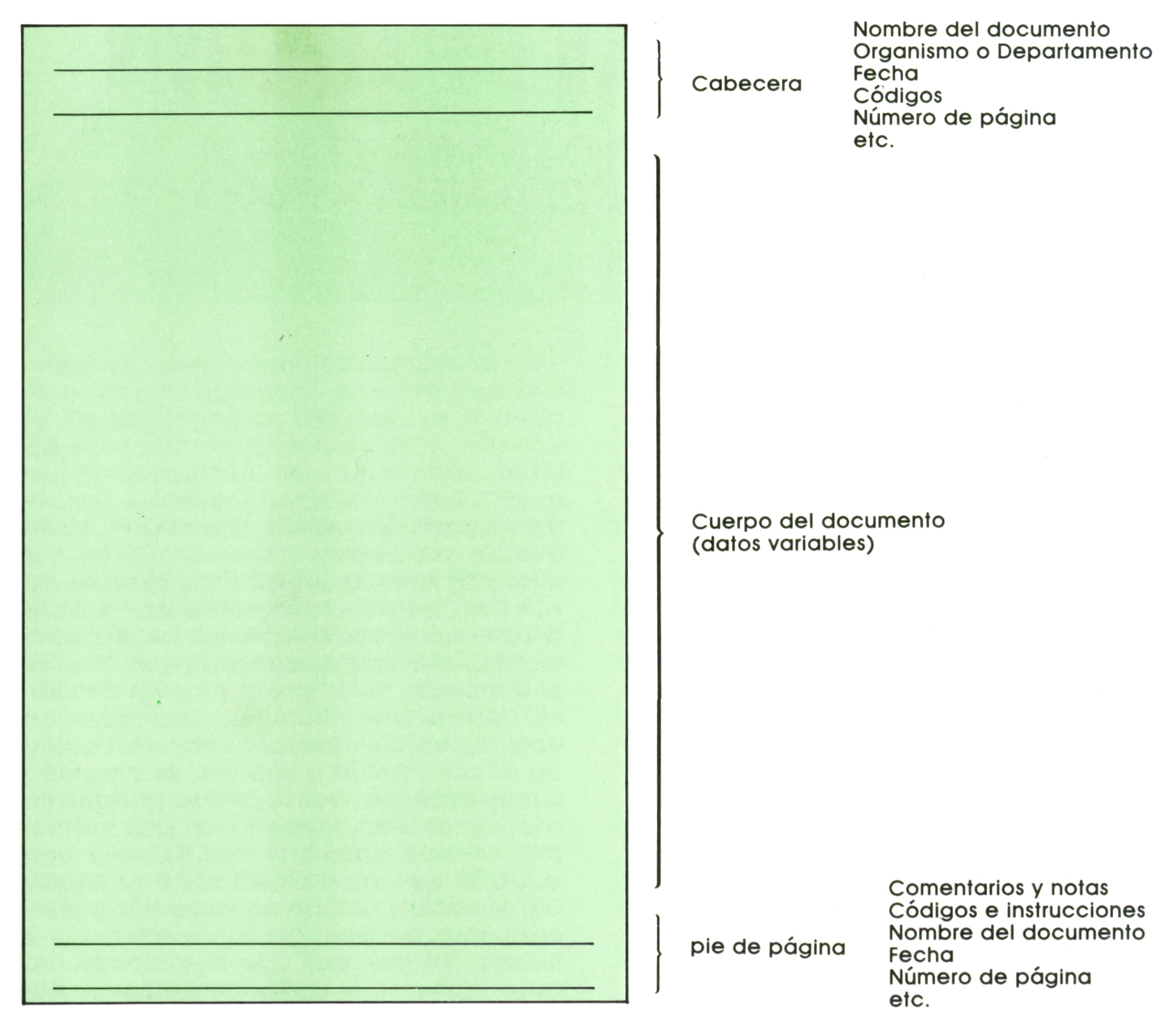
COMO conclusión de los comentarios hechos en el tomo anterior acerca de la distribución de los datos en un documento de salida no preimpreso, incluimos en la página siguiente un esquema con la distribución de las diferentes partes de que suele constar este tipo de documento.

Además del formato general y de la presentación, tamaño, distribución, etc., de las informaciones en un documento cualquiera de salida, es importante tener en cuenta algunas consideraciones respecto de los datos que se incluyen en él y cómo han de ser elaborados:

a) La información ha de ser **muy seleccionada.** Es importante no incluir en el documento ni un solo dato más que los estrictamente necesarios. En ocasiones es útil, incluso, realizar algún tipo de comprobación posterior para verificar la validez y el uso que se hace de cada información impresa (a veces, mediante «encuestas» posteriores; en otros casos, mediante algún «resguardo» que se incluya en el propio impreso —como se indica más adelante—, etc.).

b) Además, en el diseño del documento hay que tener en cuenta el **orden en que se incluyen los datos.** No sólo hay que presentarlos de un modo lógico para el usuario sino del modo que sea más adecuado para el proceso que debe generarlos. En efecto, en algún caso puede parecer más claro presentar los datos generales (de totales o de integración de los datos parciales) antes que los de detalle, pero eso obliga a ir guardando todos los datos mientras se van procesando, hasta obtener los totales, para entonces confeccionar el documento de salida con los datos generales al comienzo del impreso y los de desglose detrás. Otra situación conflictiva en este aspecto se presenta también cuando el documento a imprimir es pequeño o muy estrecho, con lo que se pueden incluir varios ejemplares a lo ancho del papel: en este caso hay que estudiar una solución de compromiso entre el ahorro del espacio y tiempo de impresión y el incremento de ocupación de memoria y tiempo de proceso que se produce. Un caso típico es el de la impresión de etiquetas (para envíos postales —mailing—), aunque en este caso la escasa información incluida en cada etiqueta aconseja siempre incluir varias etiquetas simultáneamente (escribiendo sobre toda una línea de impresión a todo el ancho del papel).

c) Se deben **incluir rótulos** o encabezamientos para todos los campos. Si el documento se edita sobre un papel preimpreso la etiqueta puede ser más descriptiva del contenido del campo e incluir códigos, notas, etc. (utilizando, incluso, tipos de letras menores). Si los encabezamientos de los campos han de ser escritos por la impresora hay que condensarlos lo más posible, dentro de que mantengan la claridad necesaria del contenido. En este sentido, hay que tener especial cuidado con las **abreviaturas** que pueden resultar sumamente confu-

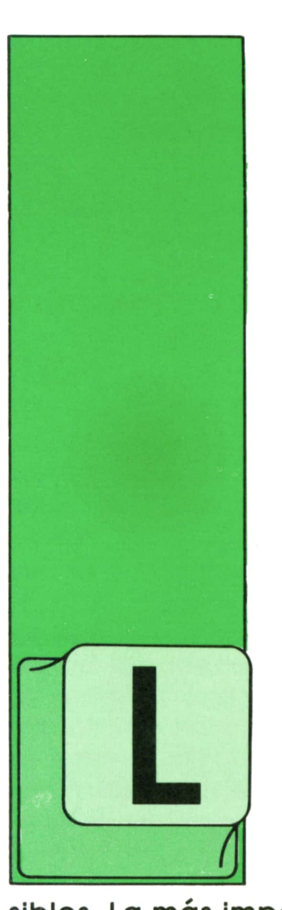


sas para quien no conozca en detalle el contenido del documento y los conceptos que en él se incluyen.

d) **Describirse los diferentes conceptos utilizados.** Esta norma, que enlaza con lo que se ha comentado de las abreviaturas, se refiere también a las **unidades** empleadas en el documento. Es importante diseñar los impresos para que puedan ser utilizados por personas «no introducidas». Es humano que el analista considere obvio saber en qué unidades se mide cada cantidad impresa, porque él conoce a fondo la aplicación de que se trata, pero el profesional experimentado sabe cuánto «oscurece» un documento la falta de información sobre las unidades, en algunos impresos complejos. Además es conveniente, en la medida de lo posible y según el espacio de que se disponga, dar el significado de los diferentes **códigos** empleados en el documento.

e) Debe estudiarse cuidadosamente el **número de copias** a imprimir. Los valores máximos previstos por los fabricantes de impresoras varían entre tres y cinco copias, pero realmente utilizando los papeles autocopiativos o de calco usuales es difícil obtener más de 3 copias claras, legibles y cómodas de utilizar. En ocasiones también sucede que se imprimen copias iguales para diferentes destinos con lo que cada usuario sólo usa una parte pequeña de los datos impresos; es necesario estudiar la posibilidad de sustituir este «gran documento» con muchas copias por varios más sencillos impresos en original y copia nada más.

TECNICAS DE PROGRAMACION





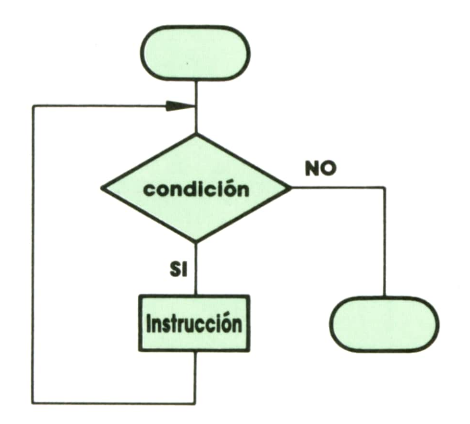
Instrucciones en bucle

LA tercera forma fundamental de las instrucciones de la programación «bien estructurada» es la instrucción en bucle, de la que también existen varias formas posibles. La más importante es la siguiente:

MIENTRAS condición HACER instrucción

es decir, «repetir una y otra vez la ejecución de la instrucción mientras se cumpla la condición. Pero, en cuanto ésta deje de cumplirse, pasar a la instrucción siguiente».

El organigrama de esta instrucción es el siguiente:



Normalmente, la condición que aparece en una instrucción MIENTRAS puede ser cualquier expresión de resultado lógico, como las que vimos en el tomo 12, que pertenecían a dos grupos principales: comparaciones entre expresiones de cualquier tipo o unión mediante funciones lógicas de dos o más expresiones lógicas cualesquiera.

En cambio, la instrucción ejecutable que aparece en la segunda parte de la instrucción de bucle puede ser de cualquier clase: una asignación, una instrucción condicional, un bloque secuencial, una llamada a subrutina, una orden de entrada o de salida... Una instrucción que pueda efectuarse de modo independiente en otro lugar del programa puede ser siempre parte de una instrucción de bucle. Unicamente se recomienda no utilizar dentro de las instrucciones de bucle las instrucciones de transferencia de control.

En particular, la instrucción ejecutable de la instrucción de bucle puede ser otra instrucción de bucle. Esto significa que las instrucciones de bucle pueden encadenarse para formar estructuras más complicadas.

Hay que tener un cuidado especial al utilizar las instrucciones de bucle, pues con ellas es posible conseguir que nuestro programa entre en una situación de bucle cerrado permanente, de la que no pueda salir más que presionando la tecla de interrupción de programa (si existe), la tecla RESET o su equivalente, o incluso apagando el ordenador. En efecto, podría suceder que la condición de la instrucción de bucle fuera siempre verdadera, y no resultara afectada por la instrucción ejecutable del interior del bucle. Si eso ocurre, la instrucción se ejecutará una vez y otra, continuamente, sin terminar jamás. Cuando sucede esto, significa que la instrucción de bucle está mal programada, pues siempre debe es-

tar construida de tal manera que sea posible que la instrucción interior ejecutable cambie la situación y que la condición de continuación deje de cumplirse.

Veamos un ejemplo de instrucción de bucle mal programada:

MIENTRAS X = 0 HACER Y = 1

Si se llega a esta instrucción cuando X es distinto de cero, no pasa nada. La instrucción interior del bucle no se ejecuta ni una sola vez (pues la condición no se cumplía desde el principio) y la ejecución continúa con la instrucción siguiente. Sin embargo, si se llega a esta instrucción cuando el valor de X es igual a cero, la condición se cumple, por lo que la instrucción interior comienza a ejecutarse repetidamente hasta que la condición deje de cumplirse, lo que no ocurrirá jamás, pues la instrucción interior no modifica el valor de X, que seguirá siendo cero indefinidamente.

En cambio, la siguiente instrucción estaría bien construida:

MIENTRAS X>0 HACER X = X − 1

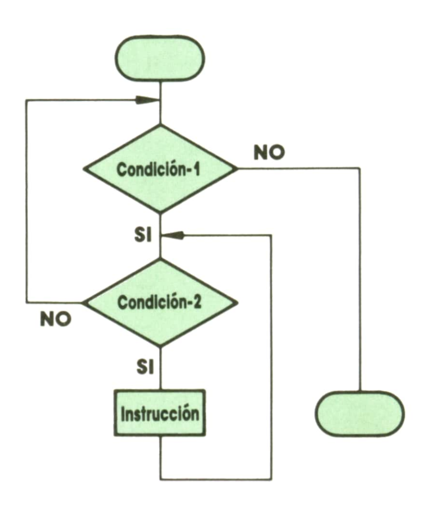
En efecto, si se llega a ella con X cero o negativo, la instrucción no se ejecutará, pues la condición no se cumple. En cambio, si se llega a ella con X positivo, la condición se cumple y la instrucción interior se ejecutará repetidamente. Pero como esta instrucción cambia el valor de X, reduciéndolo en una unidad cada vez que se ejecuta, más pronto o más tarde este valor llegará a ser cero o negativo, y en ese momento el bucle dejará de ejecutarse.

Téngase en cuenta, sin embargo, que si el valor inicial que X es muy grande (por ejemplo, un millón) el bucle se ejecutará tantas veces que nuestro programa tardará muchísimo tiempo en terminar, lo que significa que no será muy útil.

Veamos algunos ejemplos más de instrucciones de bucle:

MIENTRAS condición-1 HACER
MIENTRAS condición-2 HACER
instrucción

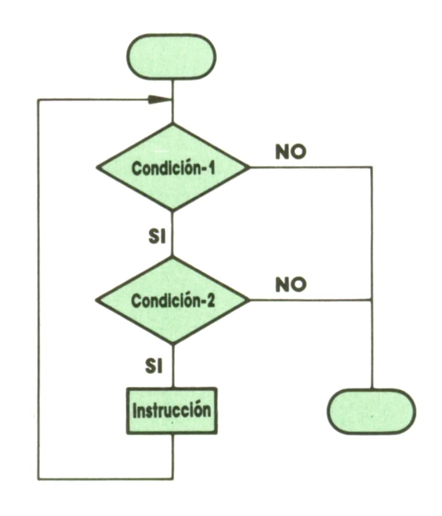
cuyo organigrama es:



Tenemos aquí una instrucción de bucle cuya instrucción interior es otra instrucción de bucle. ¡Cuidado con ella! Puede meter nuestro programa en un bucle cerrado permanente, que no tenga salida, aun cuando la instrucción interior no llegue a ejecutarse jamás. Obsérvese el organigrama. En un momento determinado, puede ser que no se cumpla la condición-2, pero sí se cumpla la condición-1. En tal caso, el primer bucle seguirá ejecutándose indefinidamente, mientras el bucle más interno no hará nunca nada, con lo que la situación de las condiciones no se modificará y el programa no podrá continuar jamás con la instrucción siguiente. Si existe este peligro, es mejor programar el bucle de la siguiente manera:

MIENTRAS condición-1 y condición-2
HACER instrucción

cuyo organigrama es:



Obsérvese que, puesta de esta forma, la instrucción no puede entrar en bucle cerrado permanente, a menos que ambas condiciones sean siempre verdaderas.

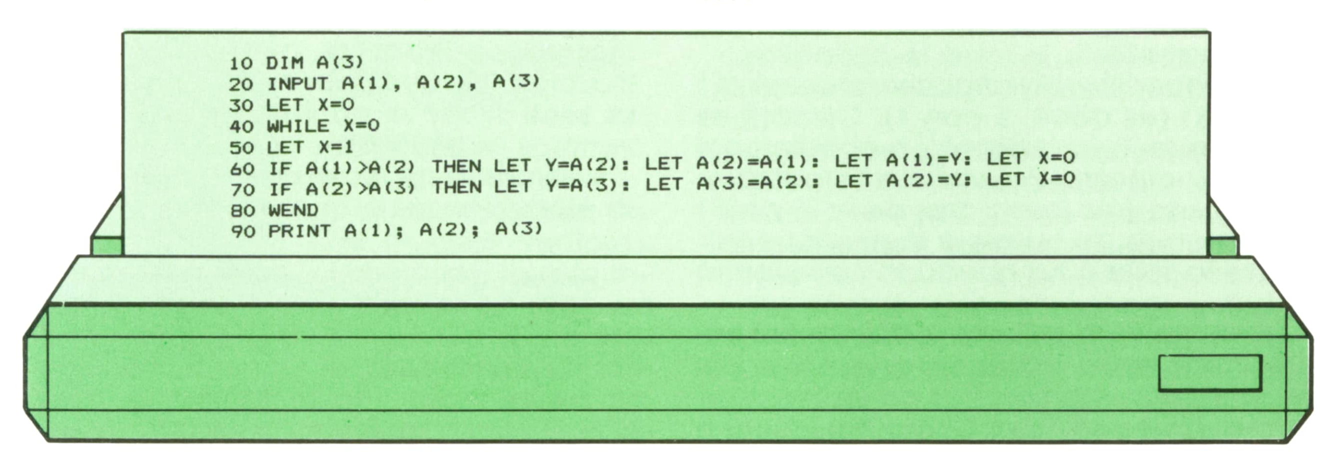
Veamos ahora cómo se programan las instrucciones de bucle de este tipo en los tres lenguajes que estamos utilizando como elemento de comparación: BASIC, PASCAL Y APL.

No todos los intérpretes de BASIC aceptan esta forma de la instrucción de bucle, pero hay muchos que sí la reconocen, como el BASIC de Microsoft o el que utiliza el IBM PC. En estos casos, la instrucción se descompone en la siguiente secuencia:

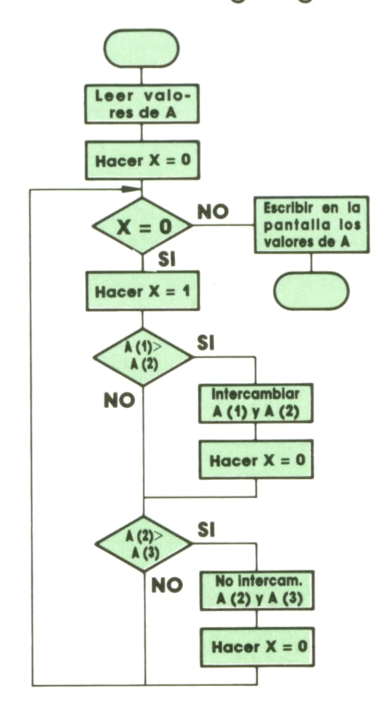
1. Una instrucción de la forma «WHILE condición», donde WHILE es la palabra inglesa que significa MIENTRAS.
2. Una o varias instrucciones ejecutables, que corresponden a la instrucción interior del bucle descrita más arriba.
3. La instrucción WEND, que señala el fin de la instrucción del bucle.

Cada una de las instrucciones indicadas puede tener su propio número de instrucción.

Veamos un ejemplllo de un programa BASIC que utiliza una instrucción de bucle:



Este programa lee tres valores numéricos cualesquiera sobre los elementos de la serie A y los ordena de menor a mayor, utilizando una instrucción de bucle. Veamos cuál es su organigrama:



Veamos cómo funciona la instrucción de bucle, que aquí aparece dividida en cinco, que llevan los números 40 a 80:

Inicialmente, la variable X tiene el valor cero (instrucción 30), por lo que la instrucción de bucle se ejecutará al menos una vez. Lo primero que se hace en el interior del bucle es asignarle a X el valor 1. De esta manera, si las dos instrucciones condicionales que siguen (con los números 60 y 70) no hicieran nada, la condición del bucle dejaría de cumplirse, por lo que se pondría punto final al programa.

La primera instrucción condicional compara los elementos 1 y 2 de la serie A. Si el primero es mayor, intercambia sus valores (y señala, en la variable X, que el bucle debe ejecutarse al menos una vez más). En caso contrario, no hace nada. La segunda instrucción condicional hace exactamente lo mismo con los elementos 2 y 3 de la serie A.

Veamos un ejemplo. Supongamos que le proporcionamos al programa la serie de valores 3, 2, 1. La instrucción 20 los

leerá del teclado y los asignará a la variable A en ese orden. Por tanto, la serie A valdrá en ese momento 3, 2, 1.

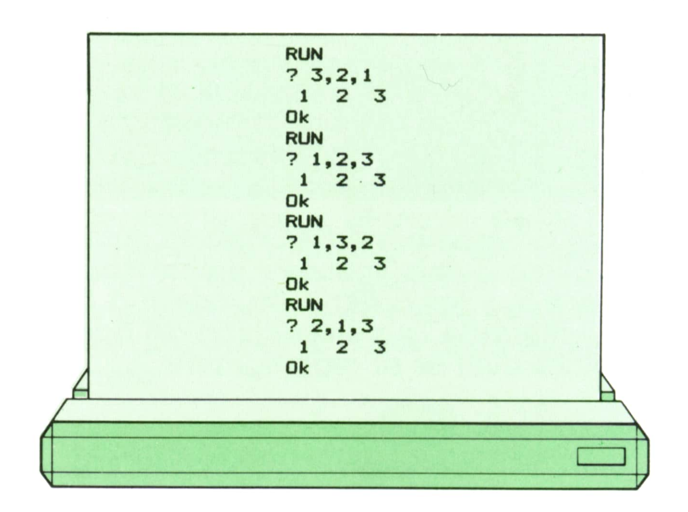
La instrucción 30 asigna a X el valor 0. La instrucción 40 comprueba que X tiene el valor cero. Como esto es cierto, comienza la ejecución del interior del bucle. En primer lugar, la instrucción 50 asigna a X el valor 1. Después, la instrucción 60 compara A(1) con A(2) (es decir, 3 con 2). Como 3 es mayor que 2, la instrucción condicional debe ejecutarse. Por tanto, intercambiamos los valores de A(1) y A(2) (obsérvese que lo hacemos a través de una variable auxiliar y, que sólo sirve para esto) y asignamos a X el valor cero. Ahora la serie A ha quedado en la forma 2, 3, 1.

La instrucción 70 compara ahora A (2) con A(3) (es decir, 3 con 1). Como 3 es mayor que 1, la instrucción condicional debe ejecutarse. Por tanto, intercambiamos igualmente los valores de A(2) y A(3) y asignamos de nuevo a X el valor cero. Ahora la serie A ha quedado en la forma 2, 1, 3.

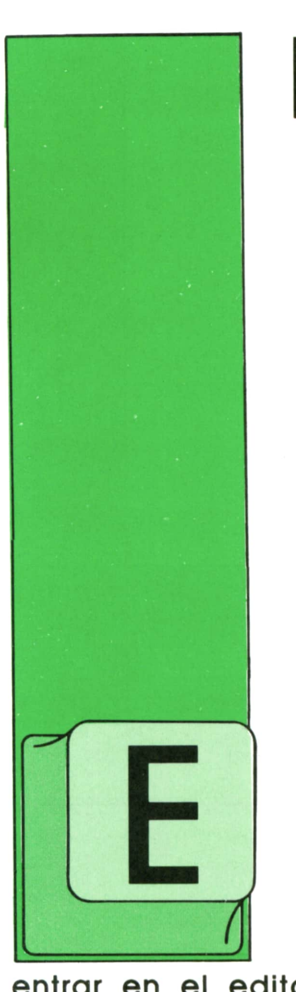
La instrucción 80 señala el final del bucle. Por tanto, debemos regresar a la instrucción 40 y comprobar de nuevo si se cumple la condición. Como X es igual a cero, es preciso ejecutar el bucle por segunda vez. Por tanto, pasamos a la instrucción 50, que asigna a X el valor 1. Después, la instrucción 60 compara A(1) con A(2) (es decir, 2 con 1). Como 2 es mayor que 1, la instrucción condicional debe ejecutarse. Por tanto, intercambiamos los valores de A(1) y A(2) y asignamos a X el valor cero. Ahora la serie A ha quedado en la forma 1, 2, 3.

La instrucción 70 compara ahora A(2) con A(3) (es decir, 2 con 3). Como 2 no es mayor que 3, la instrucción condicional no debe ejecutarse. Por tanto, pasamos a la instrucción 80, que nos envía de nuevo al principio del bucle (instrucción 40). De nuevo llegamos aquí con X igual a cero, por lo que el bucle debe efectuarse por tercera vez. La instrucción 50 asigna a X el valor 1. Después, la instrucción 60 compara A(1) con A(2) (es decir, 1 con 2). Como 1 no es mayor que 2, la instrucción condicional no debe ejecutarse. Por tanto, pasamos a la instrucción 70, que compara A(2) con A(3) (es decir, 2 con 3). Como 2 no es mayor que 3, la instrucción condicional no debe ejecutarse. Por tanto, pasamos a la instrucción 80, que nos envía de nuevo al principio del bucle (instrucción 40). Pero ahora llegamos aquí con X igual a uno, por lo que la condición del bucle no se cumple y su interior no debe ejecutarse más veces. Pasamos, por tanto, a la instrucción 90, que imprime los valores de la serie A en la pantalla, para que los veamos ordenados.

Veamos el aspecto de varios casos de la ejecución de este programa:



Obsérvese que, cualquiera que sea el orden en que demos los elementos de A, el programa los ordena correctamente, de modo que siempre terminamos con los valores 1, 2, 3.



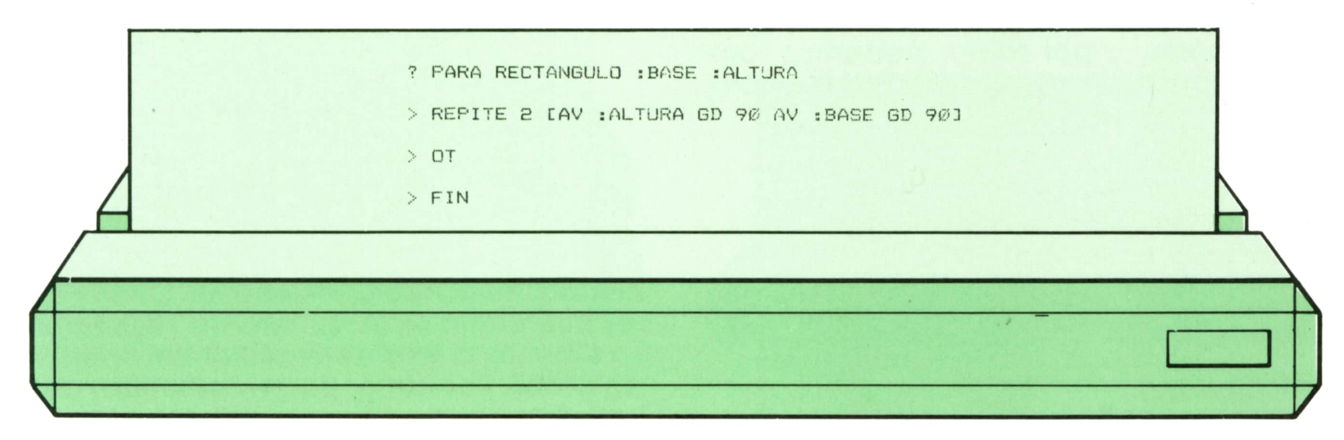
LOGO

Más dibujos usando variables

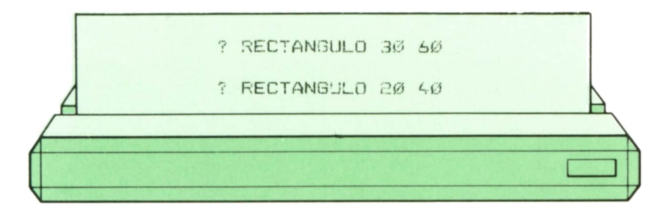
Al utilizar variables tiene muchas ventajas. Entre ellas, la más importante es que, en general, con un solo procedimiento podemos hacer varias cosas sin necesidad de entrar en el editor, sino simplemente cambiando el valor que la tortuga ha de guardar en el cajón correspondiente a la variable.

Hasta ahora hemos usado un sola variable dentro de un procedimiento. Pero esto no tiene por qué ser así, es decir, podemos añadir tantos cajones como necesitemos.

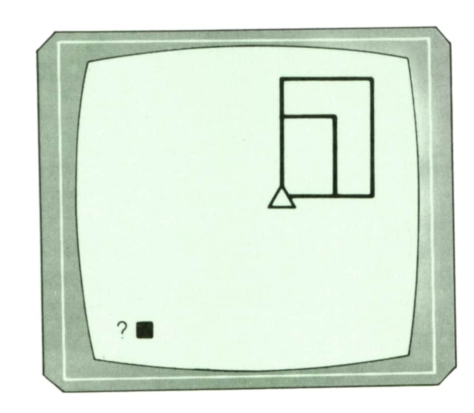
Supongamos que queremos dibujar rectángulos de tamaño variable. Como sabemos, un rectángulo tiene los lados iguales dos a dos. Por ello, hemos de tener dos variables. El procedimiento quedaría así:



Si ahora escribimos:



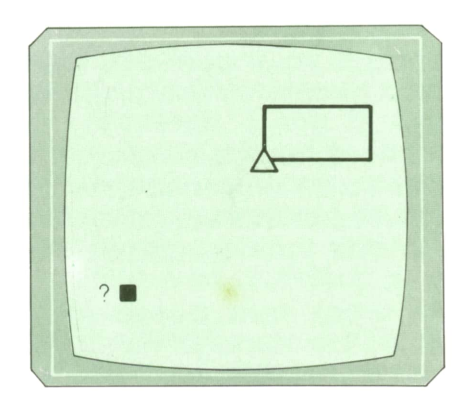
nos aparecería:



Si queremos que los rectángulos sean apaisados, no tenemos que cambiar nada en el procedimiento, porque hemos usado variables. Nos basta con poner en el cajón "BASE un valor más grande que en el cajón "ALTURA. Por ejemplo, así:



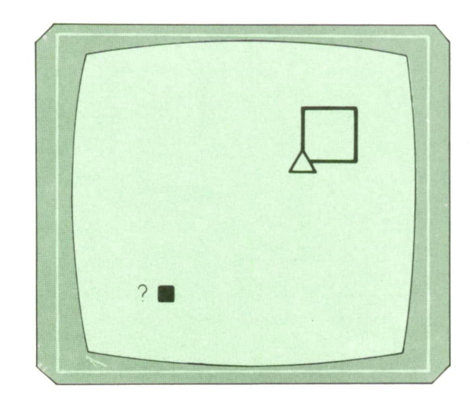
nos saldría:

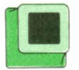


Por último, si lo que hacemos es meter el mismo valor en ambos cajones, significa que todos los lados del rectángulo son iguales, y por tanto, podemos también dibujar un cuadrado. Por ejemplo, con



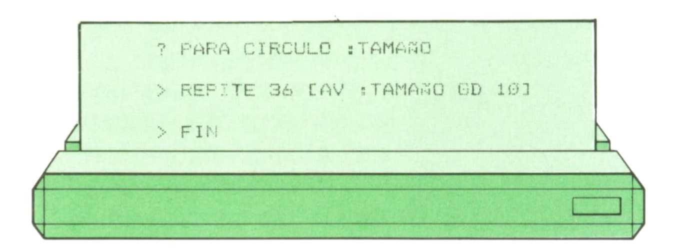
nos quedaría:





Otro procedimiento para hacer varias cosas

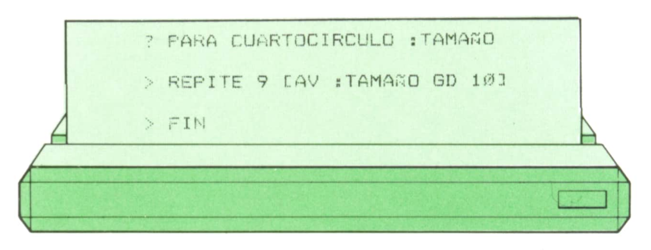
Para lograr con un solo procedimiento que la tortuga dibuje círculos de distinto tamaño, nos basta con utilizar una variable:



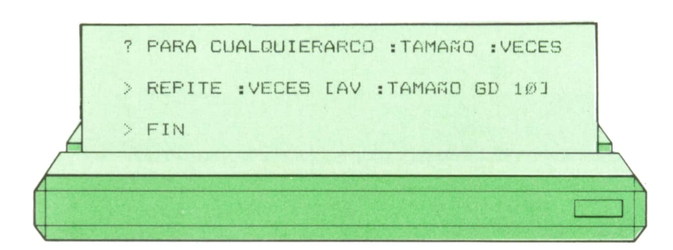
Si ahora queremos dibujar semicírculos, tendríamos el siguiente procedimiento:



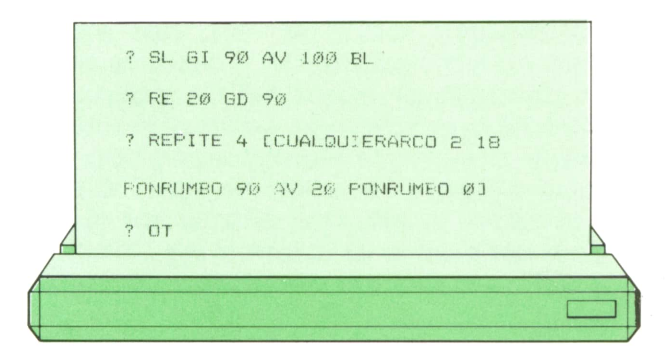
Y si también queremos obtener cuartos de circunferencia, escribiríamos:



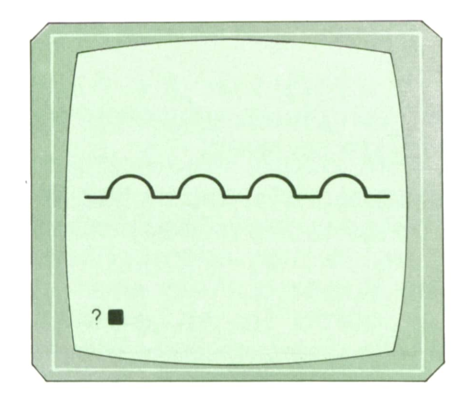
Como podemos ver, estos tres procedimientos se parecen mucho. Lo único que les diferencia es el número de veces que se ejecuta la lista de órdenes del comando REPITE. Por tanto, podemos utilizar una variable donde guardemos este número y de esta forma, usar un solo procedimiento para dibujar círculos completos, semicírculos y cuartos y arcos de circunferencia. Para ello, pondríamos:



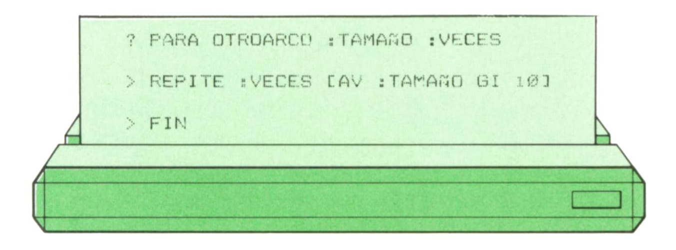
Vamos a hacer algo con él. Por ejemplo, si ahora escribimos:



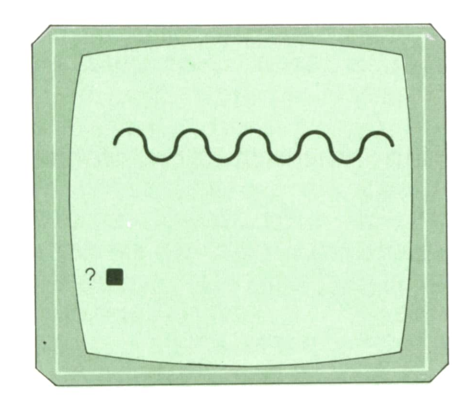
nos queda esta figura:



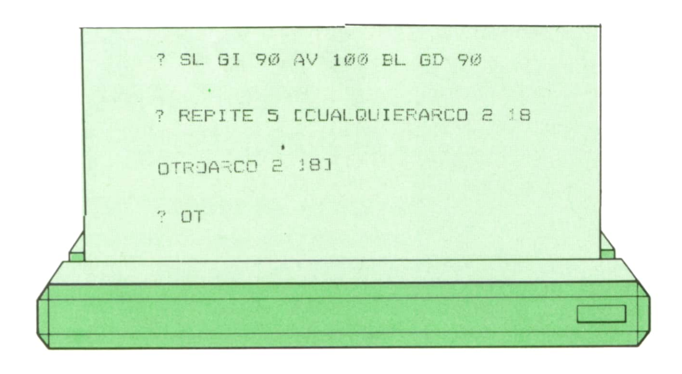
Si nos definimos el siguiente procedimiento:



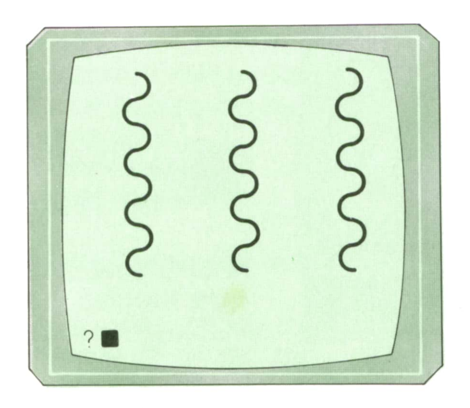
podemos dibujar una línea ondulada



así:



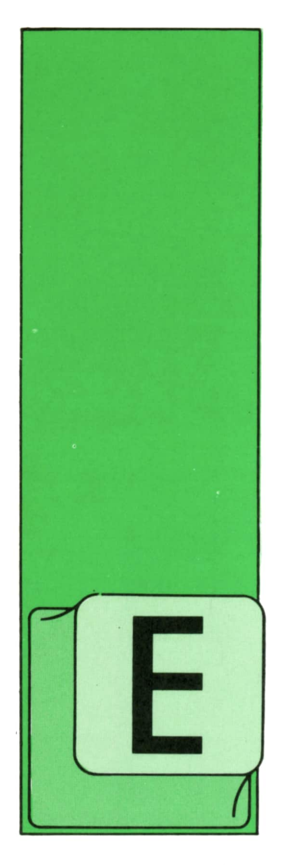
Ahora podemos hacer varias verticales:



Hay que escribir:

```
? SL PONPOS [-8Ø 7Ø] BL GD 9Ø
? REPITE 4 [CUALQUIERARCO 2 18
OTROARCO 2 18]
? SL PONPOS [Ø 7Ø] BL
? REPITE 4 [CUALQUIERARCO 2 18
OTROARCO 2 18]
? SL PONPOS [8Ø 7Ø] BL
? REPITE 4 [CUALQUIERARCO 2 18
OTROARCO 2 18]
? OT
```

PASCAL





Longitud de un texto

N el último programa que hicimos, AnalizaVarios, utilizábamos la función Longitud para determinar el tamaño real de un texto.

Probablemente el lector habrá pensado que sería mucho más sencillo escribirlo así:

```
function  Longitud (T: Texto_t): Indice_t;
(* Devuelve la longitud real del texto T *)
 var I: integer;
begin
 I:= MaxLetras;
 (* Se explora el texto de atrás para adelante *)
 (* hasta encontrar el primer carácter válido: *)
 while (T[I] = ' ') or (T[I] = chr(0)) do I:= I-1;
 Longitud:= I
end;
```

En efecto, esta manera es, en principio mucho mejor, pero tiene un inconveniente: ¿qué pasaría si todos los caracteres fueran espacios en blanco?: el índice I iría disminuyendo el valor hasta llegar a 1, tras lo cual se comprobaría el valor de T(1) y se decrementaría I nuevamente al ser aquél un espacio en blanco, por lo que acto seguido se intentaría mirar el valor de T(0), que no existe.

Dependiendo del compilador y de las opciones de compilación utilizadas (véase nota más adelante), en casos así puede que el programa se detenga dando un mensaje de error o que simplemente se ponga a buscar en zonas de memoria no correspondientes a T, con lo que sería probable que se detuviera antes o después del proceso con I teniendo un valor negativo y, por tanto, erróneo. Si es

este último nuestro caso, podría ser admisible en alguna ocasión, aunque sería sin duda una «chapuza».

Las posibles condiciones de salida del bucle son dos, el haber explorado ya todo el texto y el hallazgo de un carácter válido; sin embargo, la segunda condición sólo debe comprobarse tras verificar que no se cumple la primera, y de ahí la dificultad en hacer el proceso con un bucle tan simple (estudie el lector la posibilidad de arreglar el problema modificando la condición del bucle).

NOTAS

— En algunos lenguajes como el Modula-2, sí es posible escribir

```
while (I > 0) and ((T[I] = ' ') or
(T[I] = chr (0))) do ...
```

En una operación lógica AND sólo se evalúa la segunda parte si la primera ha resultado ser TRUE, pues si ha sido FALSE ya se sabe que ése será el resultado final y no tiene sentido seguir adelante; análogamente, en las operaciones OR sólo se evalúa la segunda parte si la primera es FALSE, pues en caso contrario el resultado ya está decidido. Esto resulta ventajoso no sólo por simplificar la exploración de tablas, sino también por el ahorro de tiempo de cálculo que puede obtenerse.

— Lo de las «opciones de compilación» consiste en que, normalmente, al compilar es posible escoger una serie de opciones como, por ejemplo, que cada vez que se haga referencia a un elemento de una tabla se verifique o no que el índice es lícito; otra opción típica es la de que se verifique automáticamente o no si hay sitio suficiente en memoria para las variables locales cada vez que se llame a un subprograma.



Elecciones a la carta

Prácticamente cualquier persona con un mínimo de experiencia utilizando ordenadores se habrá encontrado en más de una ocasión con programas que emplean los llamados «menús» de opciones. La estructura de un programa semejante podría ser:

```
Repetir
  Borrar la pantalla.
  Presentar menú.
  Preguntar.
  Leer la opción deseada.
  Según que la opción sea...
    1, hacer esto.
    2, hacer lo otro.
    3, hacer aquello.
    ...
... hasta que la opción sea acabar.
```

A su vez, los subprogramas «hacer «esto», «hacer lo otro», etc., podrían utilizar también el método del menú, en cuyo caso tendrían una estructura similar. A modo de ejemplo, vamos a hacer un programa de tipo «calculadora» que, presentando un menú de opciones en pantalla, nos permita hacer diferentes operaciones con números. Para tener mayor libertad y precisión, los números empleados serán reales; como los números reales se pueden representar de muchas maneras distintas, incluiremos la posibilidad de seleccionar cómo mostrarlos.

Siguiendo la estructura antes mencionada, el programa principal podría ser algo así:

```
program Calculadora;

var
 Formato: (Exponencial, EspacioFijo, ComaFija);
 Espacios,
 Decimales: integer;
 Opcion: char;

(**************************************************************************)
(******  Aqui falta parte del programa, que se explica aparte ******)
(**************************************************************************)

(*----------------------------------------------------------------------*)
(*                    Petición de datos.                                *)
(*----------------------------------------------------------------------*)
```

```
function PideNumero: real;
  var R: real;
 begin
  write ('Número: ');
  readln (R);
  PideNumero:= R
 end;

(*-----------------------------------------------------------------*)
(*                     Programa principal.                           *)
(*-----------------------------------------------------------------*)
begin
 Formato:= Exponencial;   (* Formato inicial *)

 repeat
  ClrScr; (* o PAGE, o lo necesario para borrar la pantalla *)

  writeln ('1 - Sumar dos números.');
  writeln ('2 - Restar dos números.');
  writeln ('3 - Multiplicar dos números.');
  writeln ('4 - Dividir dos números.');
  writeln ('5 - Calcular raíz cuadrada.');
  writeln ('6 - Variar la forma de mostrar números.');
  writeln;
  writeln ('0 - Acabar.');
  writeln;
  write ('Escoja opción:  ');
  readln (Opcion);
  writeln;
  (*----------------------------------------*)
  case Opcion of

     '1': Muestra (PideNumero + PideNumero);

     '2': Muestra (PideNumero - PideNumero);

     '3': Muestra (PideNumero * PideNumero);

     '4': Muestra (PideNumero / PideNumero);

     '5': Muestra (sqrt (abs (PideNumero)));

     '6': CambiarAspecto;

     '0': (* no hacer nada, instrucción "nula" *) ;

     else  write ('Opción no válida.')

   end; (* Fin de CASE *)
  (*----------------------------------------*)
   if Opcion <> '0' then
     begin
      writeln;
      writeln ('Pulse Intro.');
      (* Aquí se para hasta pulsar Intro: *)
      readln
     end

  until Opcion = '0'
 end.
```

Se ha utilizado la función PideNumero, que devuelve el número tecleado, directamente al hacer las operaciones, y éstas a su vez se han escrito en las propias instrucciones de llamada a Muestra, por lo que no ha resultado necesario utilizar ninguna variable de tipo Real en el programa principal.

Faltan por definir los procedimientos Muestra y CambiarAspecto. El primero se emplea para enseñar el resultado de la operación de acuerdo con el formato deseado, que viene determinado por el valor de la variable escalar Formato. Por otra parte, CambiarAspecto debe dar la posibilidad de cambiar el formato y de escoger los parámetros de presentación necesarios en algunos casos; este subprograma será también del tipo menú:

```
(*---------------------------------------------------------------*)
(*     Procedimiento para el cambio de aspecto de los números.    *)
(*---------------------------------------------------------------*)
procedure CambiarAspecto;
  var Forma: char;
  (*----------------------------------------------*)
   procedure LeerEspacios;
    begin
     writeln;
     write ('Espacios a ocupar por número: ');
     readln (Espacios)
    end;
  (*----------------------------------------------*)
begin
  ClrScr;
  writeln ('1 - Notación exponencial.');
  writeln ('2 - Coma flotante.');
  writeln ('3 - Coma fija.');
  writeln;
  write ('Escoja formato:  ');
  readln (Forma);
  case Forma of

     '1': Formato:= Exponencial;

     '2': begin
           Formato:= EspacioFijo;
           LeerEspacios
          end;

     '3': begin
           Formato:= ComaFija;
           LeerEspacios;
           writeln;
           write ('Número de decimales: ');
           readln (Decimales)
          end

     else
       begin
        writeln;
        writeln ('Formato incorrecto.')
       end

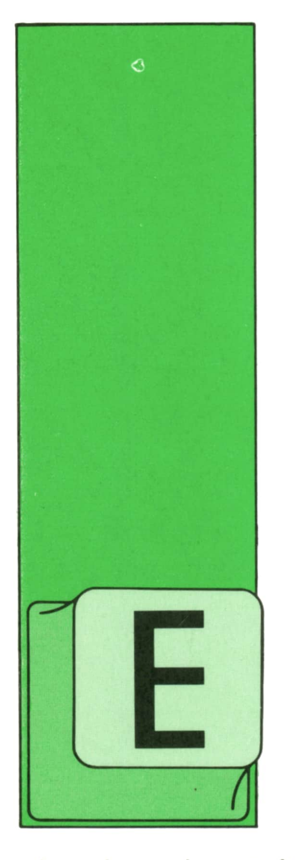
     end; (* Fin de CASE *)

end;

(*---------------------------------------------------------------*)
(*    Procedimiento de presentación de números según formato.     *)
(*---------------------------------------------------------------*)
procedure Muestra (R: real);
 begin
  write ('Resultado = ');
  case Formato of
    Exponencial : writeln (R);
    EspacioFijo : writeln (R : Espacios);
    ComaFija    : writeln (R : Espacios : Decimales)
   end
 end;
```

Lo único que tiene que hacer el lector para utilizar el programa es integrar estos dos procedimientos en el sitio correspondiente de la parte principal, modificar en su caso las instrucciones de borrado de pantalla y realizar el proceso necesario para su puesta en marcha.

OTROS LENGUAJES





Sentencias condicionales simples

EN la gran mayoría de los programas deben tomarse decisiones en función de determinados datos.

Los programas tienen la capacidad de formular «preguntas» al ordenador, o lo que es lo mismo: se realiza una acción dependiendo si la respuesta del ordenador a una pregunta es verdadera o falsa.

```
IF condición
      SENTENCIA
ELSE
      SENTENCIA
```

Si la condición que se presenta es cierta, se ejecuta la sentencia o sentencias que siguen al IF, hasta encontrar un ELSE o un punto. Si la condición es falsa y existe la cláusula ELSE, se ejecutan las instrucciones que le siguen hasta el primer punto.

Las condiciones tienen el siguiente formato:

```
literal-1 }         { = }   { literal-2 }
          }  NOT    { > }   {           }
campo-1   }         { < }   { campo-2   }
```

Las comparaciones que se pueden realizar son:

= Igual	NOT = Distinto
> Mayor	NOT > Menor o igual
< Menor	NOT < Mayor o igual

Como aplicación de esta sentencia, se debe realizar un programa que pida el código de empleado, el sueldo, su estado civil y el número de hijos. Si el empleado está casado tiene un suplemento en el sueldo de 5.000 pesetas y una gratificación del 5 por 100, mientras que si es soltero, la gratificación es del 2 por 100. Si el número de hijos es mayor o igual a tres, se da una prima de 500 pesetas por cada hijo.

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-IF.

ENVIRONMENT DIVISION.

DATA DIVISION.


 WORKING-STORAGE SECTION.

   01 REG-EMPLEADO.
      05 CODIGO                          PIC X(5).
      05 SUELDO                          PIC 9(6).
      05 ESTADO                          PIC X.
      05 NUM-HIJOS                       PIC 99.

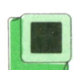
   01 GRATIF                             PIC 9.
   01 TOTAL                              PIC 9(7).
```

```
PROCEDURE DIVISION.

INICIO.
        DISPLAY 'CODIGO'.
        ACCEPT CODIGO.
        DISPLAY 'SUELDO'.
        ACCEPT SUELDO.
        DISPLAY 'ESTADO CIVIL'.
        ACCEPT ESTADO.
        DISPLAY 'NUMERO DE HIJOS'.
        ACCEPT NUM-HIJOS.
        IF ESTADO = 'C'
           ADD 5000 TO SUELDO
           MOVE 5   TO GRATIF
        ELSE
           MOVE 2   TO GRATIF.
        COMPUTE TOTAL = SUELDO + SUELDO * GRATIF / 10.
        IF NUM-HIJOS NOT < 3
           COMPUTE TOTAL = TOTAL + 500 * NUM-HIJOS.
        DISPLAY CODIGO 'DEBE COBRAR ' TOTAL.

FIN-PROGRAMA.
        STOP RUN.
```

En las instrucciones condicionales tiene mucha importancia la colocación del punto, puesto que limita la finalización del IF. Por ejemplo, si detrás de la sentencia MOVE 2 TO GRATIF, no se colocase un punto, la instrucción que le sigue (COMPUTE), sólo se ejecutaría si no se cumpliese la condición, cambiando toda la lógica del programa.

 Operadores lógicos

Las condiciones que se han visto hasta ahora son condicionales simples. Varias condiciones simples pueden unirse con operadores lógicos, formando condiciones compuestas. Los operadores que permiten realizar esta unión se muestran por orden de prioridad.

- — NOT: No.
- — AND: Y.
- — OR: O.

El funcionamiento se muestra en la siguiente figura.

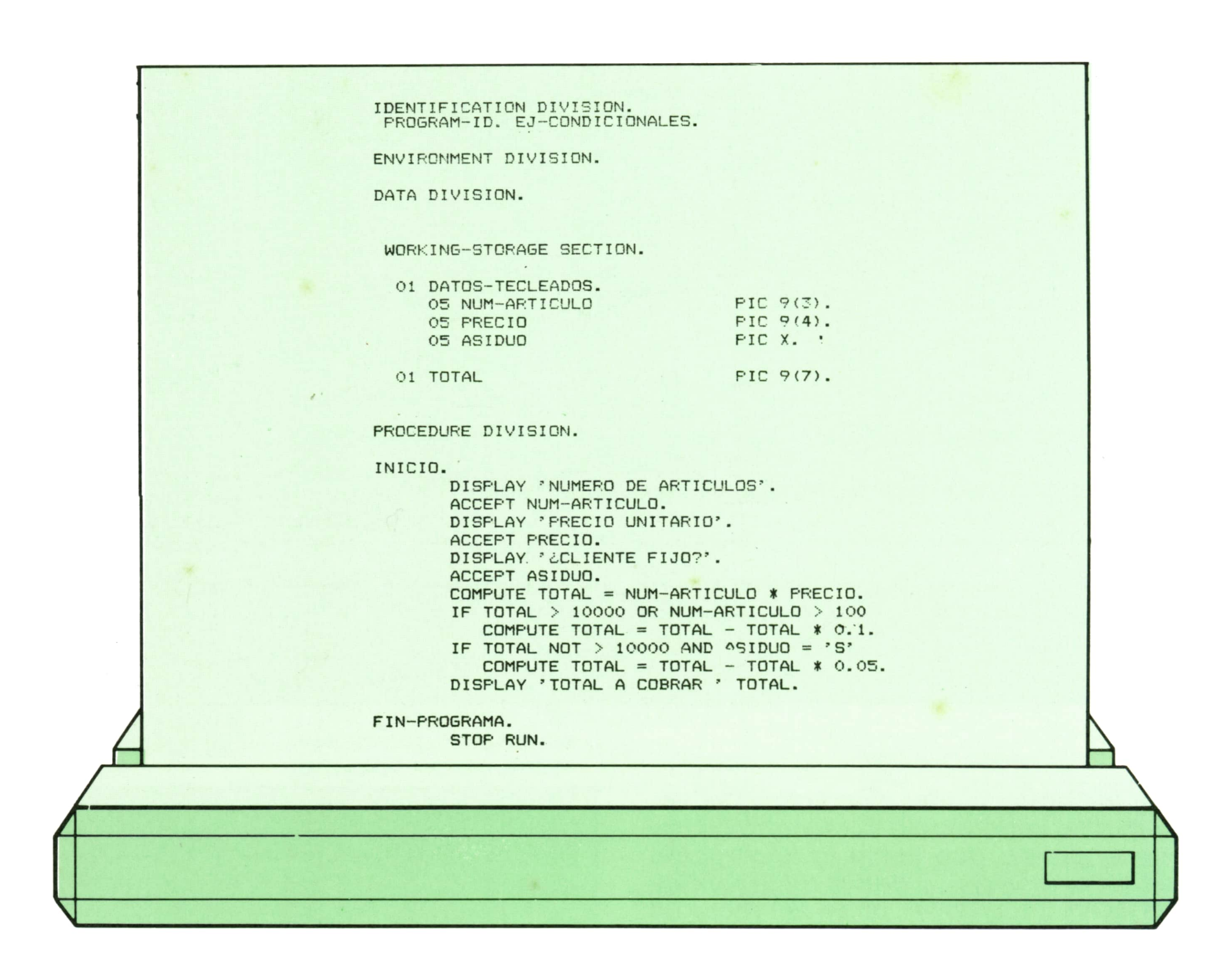
C1	C2	NOT C1	C1 AND C2	C1 OR C2
F	F	V	F	F
F	V	V	F	V
V	F	F	F	V
V	V	F	V	V

EL NOT aplicado a una condición cambia su significado. Para que una condición formada por dos condiciones unidas por un AND sea cierta han de serlo las dos condicionales simples. Por el contrario, si están unidas por un OR basta con que sea cierta una de las dos.

Determinada tienda necesita un programa para calcular el monto de sus ventas. Para ello el vendedor tecleará el número de artículos vendidos, el precio unitario de los mismos y si se trata de un cliente fijo.

Si el valor de la compra es superior a 10.000 pesetas o se han vendido más de cien artículos, se le aplica al comprador un descuento del 10 por 100. Si el cliente es habitual y su compra es mayor o igual a 1.000 pesetas, se le hace un descuento del 5 por 100.

El programa se muestra en la figura siguiente.



```
IDENTIFICATION DIVISION.
 PROGRAM-ID. EJ-CONDICIONALES.

ENVIRONMENT DIVISION.

DATA DIVISION.


 WORKING-STORAGE SECTION.

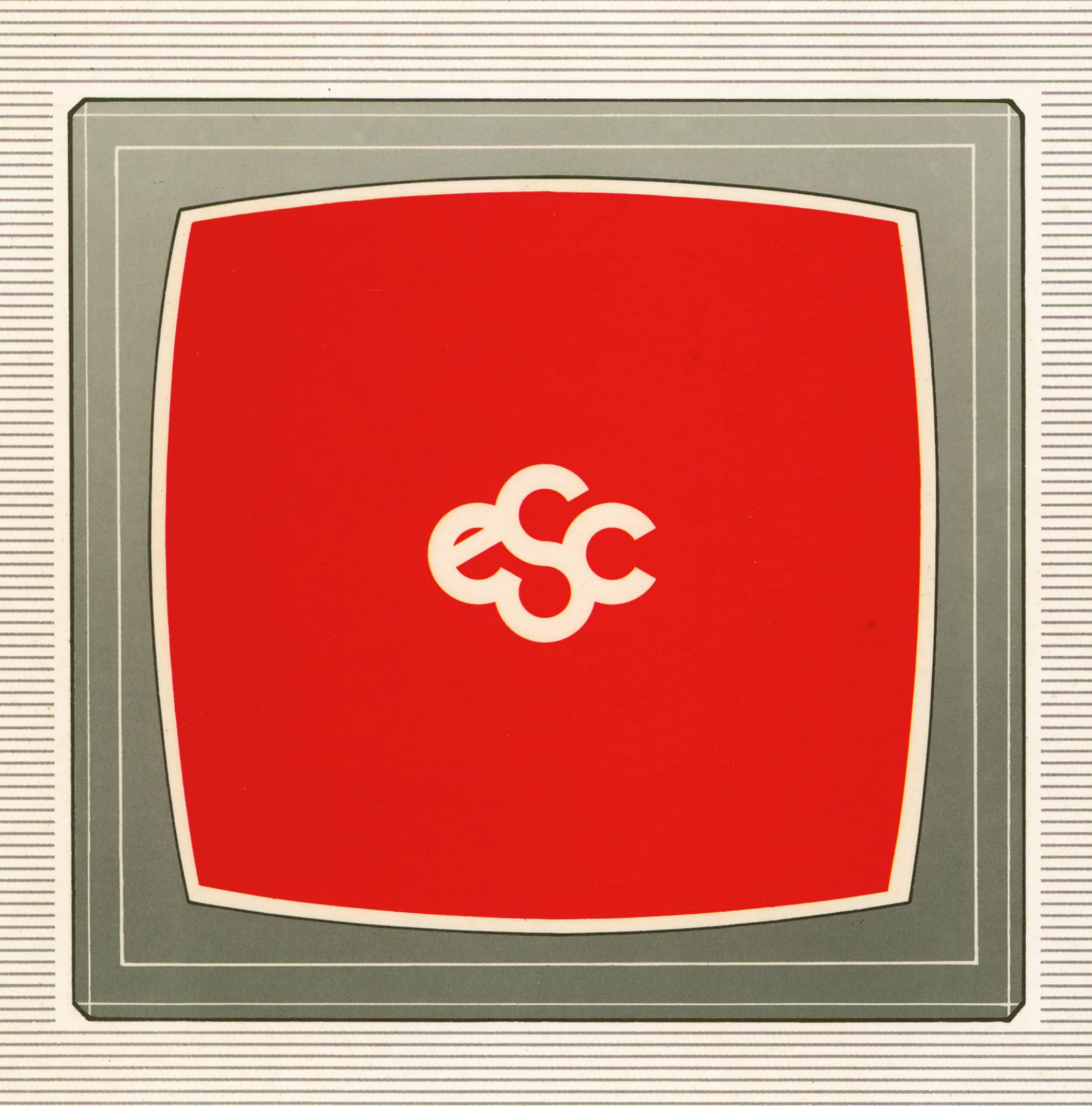
  01 DATOS-TECLEADOS.
     05 NUM-ARTICULO              PIC 9(3).
     05 PRECIO                    PIC 9(4).
     05 ASIDUO                    PIC X.

  01 TOTAL                        PIC 9(7).


PROCEDURE DIVISION.

INICIO.
       DISPLAY 'NUMERO DE ARTICULOS'.
       ACCEPT NUM-ARTICULO.
       DISPLAY 'PRECIO UNITARIO'.
       ACCEPT PRECIO.
       DISPLAY '¿CLIENTE FIJO?'.
       ACCEPT ASIDUO.
       COMPUTE TOTAL = NUM-ARTICULO * PRECIO.
       IF TOTAL > 10000 OR NUM-ARTICULO > 100
          COMPUTE TOTAL = TOTAL - TOTAL * 0.1.
       IF TOTAL NOT > 10000 AND ASIDUO = 'S'
          COMPUTE TOTAL = TOTAL - TOTAL * 0.05.
       DISPLAY 'TOTAL A COBRAR ' TOTAL.

FIN-PROGRAMA.
       STOP RUN.
```



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼